Delphi Prism e ASP.NET
Reaproveite seus conhecimentos Win32 / Desktop para criar aplicações Web – Parte 2

Clube Delphi +PHP

DevMedia

PHP
jQuery e AJAX

Ano 9 :: Edição 119 :: R$ 11,90

Boas Práticas
Design Patterns – O padrão Observer

REST e JSON

Aplicações DataSnap ricas e inteligentes

Brinde

Comprando esta edição ganhe 4 vídeo-aulas:

- Delphi e POO - Construtores, Destrutores e Propriedades
- Multitouch e Gestures
- Delphi for PHP – Acesso a banco de dados
- StarTeam – Campos customizados em Change Requests

Quick Update
Produtividade no acesso a dados - Herança de Data Modules

Gestures e Multitouch
Crie aplicações em Delphi semelhantes aos aplicativos para iPhone

ISSN 1517990-7

Dificuldades em encontrar conteúdo sobre .Net para iniciantes?

A DevMedia simplifica pra você!

easy .net magazine

Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

A Easy.Net não é feita em PDF ou Flash!
Ela utiliza um aplicativo 100% online.

DevMedia

Acesse agora a edição Nº 1 totalmente grátis e comprove

www.devmedia.com.br/easynet

Easy.Net Magazine, seu curso inicial de .Net!

Sumário

Quick Update

7 – Trabalhando com herança de Data Modules
[*Daniel Sobrinho Laporte*]

Delphi

Web – Novidades - AJAX

10 – DataSnap 2010
Crie servidores de aplicação escaláveis utilizando REST e JSON
[*Rodrigo Carreiro Mourão*]

Novidades – Boa Ideia – Tutorial

22 – Gestures e Multitouch
Crie aplicações em Delphi semelhantes aos aplicativos para iPhone
[*Álison Bissoli Dias*]

Boas Práticas

Boas Práticas

30 – Padrão Observer
Não ligue para mim. Eu ligo para você!
[*Rafael Stavarengo*]

Easy Delphi

Easy Delphi - Mini-Curso – Web – Tutorial

39 – Delphi Prism e ASP.NET
Reaproveite seus conhecimentos Win32 / Desktop para criar aplicações Web – Parte 2
[*Guinther Pauli*]

PHP

PHP - AJAX

44 – Introdução à jQuery no PHP
Conheça a biblioteca JavaScript que mais se destaca no mercado
[*Lucas Simões Maistro*]

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Brinde na web desta edição - 4 vídeos! Confira:

4 Vídeos

- **Curso Online-Novidades no Delphi 2010-Parte 15-Touch, MultiTouch e Gesture-Parte 2**
  http://www.devmedia.com.br/post-16694-Curso-Online-Novidades-no-Delphi-2010-Parte-15-Touch-MultiTouch-e-Gesture-Parte-2.html

- **Curso StarTeam – Customizando campos em Change Requests**
  http://www.devmedia.com.br/post-16544-StarTeam-Curso-basico-Parte-XV-Customizando-campos-em-Change-Requests.html

- **Curso Online-Dominando a Orientação a Objetos e Componentes em Delphi (Avançado)-Parte II-Exemplos práticos de como se trabalhar com construtores, destrutores e propriedades indexadas**
  http://www.devmedia.com.br/post-9909-Curso-Online-Dominando-a-Orientacao-a-Objetos-e-Componentes-em-Delphi--Avancado-Parte-II-Exemplos-praticos-de-como-se-trabalhar-com-construtores-destrutores-e-propriedades-indexadas.html

- **Novidades da nova versão do Delphi For Php 2.0 -Parte IV-Novos recursos relacionados a banco**
  http://www.devmedia.com.br/post-9889-Novidades-da-nova-versao-do-Delphi-For-Php-2-0--Parte-IV-Novos-recursos-relacionados-a-banco-de-dados.html

Para acessar os vídeos utilize os seguintes dados
Login: **DVM.PLS** *Senha:* **EQ0X40**

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **DVM.PLS** Senha: **EQ0X40**

Clube Delphi +PHP

Ano 9 - 119ª Edição 2010 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Paulo Quicoli
pauloquicoli@devmedia.com.br

Daniel Laporte
daniel.laporte@gmail.com

Rodrigo Carreiro Mourão
rodrigocarreiro@rmfactory.com.br

Equipe Editorial
Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

Capa e Diagramação
Romulo Araujo
romulo@devmedia.com.br

Coordenação Geral
Daniella Costa - daniella@devmedia.com.br

Revisão e Supervisão
Thiago Vincenzo - thiago.v.ciancio@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Cristiany Queiroz – Atendimento ao Leitor
www.devmedia.com.br/mancad
(21) 3382-5038

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

EDITORIAL

REST - Representational State Transfer – é o tema do momento. Vamos nos remeter aos primórdios do desenvolvimento Web com o Delphi para entender este modelo. O DataSnap suporta XML / HTTP desde a versão 6 do Delphi, sem depender do COM, quando a interface IUnknown passou a ser um alias de IInterface (com o nascimento do Kylix). Por tabela, o DataSnap Server pôde se comunicar via HTTP com outros clientes, normalmente Delphi mesmo, via Web (veja edição 70), tudo via SOAP / Web Services, o que também trouxera o DataSnap ao mundo Linux. Agora certamente são poucos os que lembram do Internet Express do Delphi 5. Essa grande tecnologia, que surgiu com essa versão do Delphi e também ficou parada por lá mesmo, foi uma das melhores implementações de arquitetura Web que já foi projetada, em minha opinião. A ideia era simples: um servidor DataSnap, através de um componente chamado XMLBroker, rodava em conjunto com um servidor Web, que poderia ser WebBroker do tipo CGI, ISAPI etc. Este componente devolvia dados de tabelas do BD em formato XML, um DataPacket, o mesmo usado em aplicações Win32. O interessante é que esse DataPacket era tratado não por um cliente MIDAS / DataSnap "thin-client", mas diretamente de código JavaScript rodando em um browser. Isso permitia que dados fossem manipulados sem fazer "n" requisições (postbacks) ao servidor. Era basicamente um ClientDataSet todo implementado em JS. Pois bem, anos se passaram e aqui estamos com uma arquitetura muito semelhante à apresentada pela Borland na época. Vou fazer um comparativo: imagine que você tenha um banco de dados com centenas de tabelas, a partir dele criará uma aplicação Delphi Win32 que irá manipular esses dados (inserir, atualizar, consultar etc.). Certamente você não iria ficar contente se eu dissesse que você não poderia usar SQL na aplicação Delphi para acessar esses dados, certo? Imagine que para cada tabela, e para cada operação (update, delete, insert) e para cada tipo de consulta (select *, select com parâmetros - where) você tivesse que criar Stored Procedures? Pois bem, no cenário que comentei, você teria que criar cerca de 10 mil SPs. Não é assim que funciona no modelo relacional, mas é (ou era) assim que funciona no modelo Web atual que precise "enriquecer" o cliente. Uma grande tendência nos aplicativos Web tradicionais (Web 2.0) é que eles requisitem dados diretamente de um servidor, usando código cliente JS. A forma mais simples de fazer isso é usando AJAX / jQuery para solicitar dados em um formato que possa ser manipulado no browser, no caso XML ou JSON (JavaScript Object Notation), via Web Services. Porém, quando se fala em Web Services, estamos falando em escrever métodos no servidor, um padrão do RPC – Remote Procedure Call (daí minha analogia com as SPs). E se agora todo o poder está com o cliente, por que não deixar ele consultar dados no servidor sem que para isso tenhamos que escrever centenas de Web Services? E se o browser pudesse fazer Selects, Updates, Deletes, Inserts? A resposta é que ele não pode, desde que se coloque um servidor REST no meio, que permita o uso de métodos padrão HTTP (como GET e POST). Aí entra em cena um novo tipo de aplicação, chamada RIA (Rich Internet Application). O servidor devolve dados em XML e JSON e os clientes consomem e descobrem esses dados sem depender de métodos, eles descobrem intuitivamente através da URI. Por exemplo, para consultar dados da tabela clientes acessaríamos algo como http://servidor.com/app/clientes (select * from clientes). Para acessar os dados do cliente de código número 1, acessaríamos http://servidor.com/app/clientes(1) (select * from clientes where ID=1). E como estamos falando de XML / JSON, que são formatos padrões do mercado e devolvidos pelo servidor, agora é possível criar uma gama muito grande de aplicações clientes usando diferentes tecnologias, como Flash, Silverlight, Delphi Win32, WPF, PHP, aplicativos para iPhone e tudo mais que entender texto. É nesse contexto que entram tecnologias como WCF Data Services do .NET Framework 3.5 / 4.0, e também o DataSnap 2010, que adotam o padrão REST, tema do artigo destaque desta edição, na matéria de Rodrigo Carreiro.

Ainda nesta edição, aprenda com o Álisson Bissoli a criar aplicativos com suporte a Multitouch, aplique boas práticas como o uso do padrão de projeto Observer, lendo o artigo do Rafael Stavarengo, reaproveite seus conhecimentos Desktop Win32 para desenvolver para Web através do artigo de minha autoria, e na sessão PHP veja no artigo do Lucas Simões como usar jQuery com PHP.

Boa leitura!

Guinther Pauli

guinther@devmedia.com.br

Certificado Delphi 3,5,6,7,2005,2006,Web,Kylix

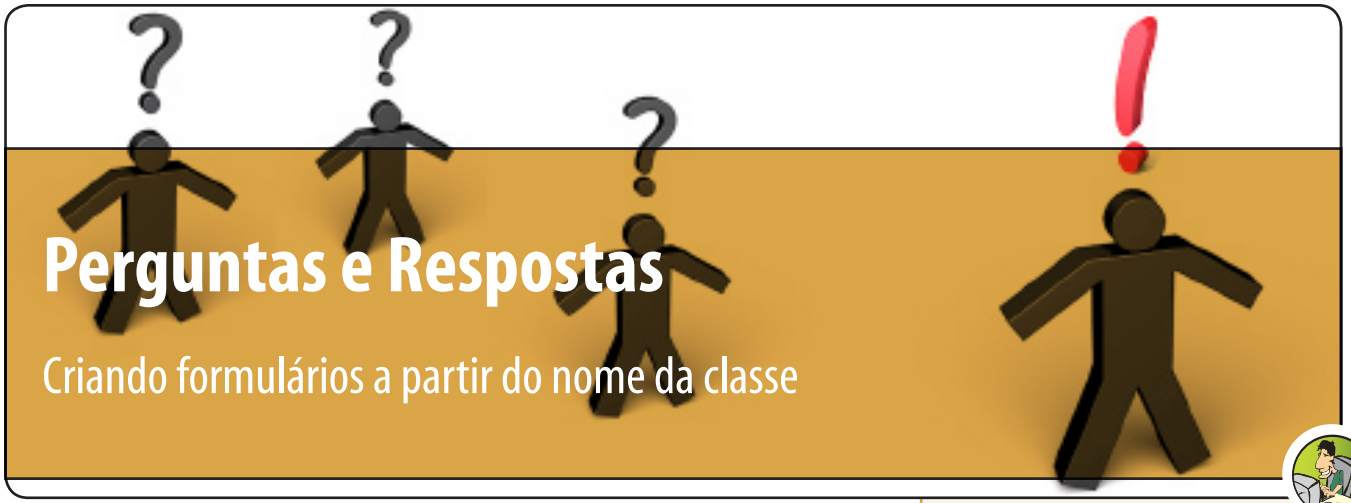
Microsoft Certified: MCP,MCAD,MCSD.NET,MCTS,MCPD (ASP.NET,C#)

guintherpauli.blogspot.com - twitter.com/guintherpauli

ClubeDelphi PLUS

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

DevMedia group



Perguntas e Respostas

Criando formulários a partir do nome da classe

Pergunta respondidas por:
Daniel Sobrinho

Gostaria de saber se é possível criar um formulário a partir do nome de sua classe (em string, não a partir do tipo ou do nome da variável). Este mês quem responde é nosso editor técnico Daniel Sobrinho.

Dúvida enviada pela leitora Renata Alves Behr - Curitiba - PR

Esta é uma maneira muito interessante de se implementar questões de segurança, ou restringir o acesso de formulários, personalizar seus aplicativos, entre tantas outras funcionalidades. Para atender a solução da nossa amiga leitora criaremos uma aplicação que demonstra como abrir e criar um formulário a partir de um nome digitado em um Edit qualquer. Para iniciar abra seu Delphi 2010 e crie uma nova aplicação VCL Forms. Adicione um formulário a sua aplicação. Nomeie o Form1 para frmPrincipal e o Form2 para frmAuxiliar, salvando suas units como untPrincipal.pas e untAuxiliar.pas respectivamente e o projeto como AskTheExpert.dproj. Insira no frmPrincipal um Edit, um Label, um CheckBox e um BitBtn. Configure o Layout da sua aplicação conforme a **Figura 1**.

Figura 1. Layout da aplicação

Antes de mais nada adicione o frmAuxiliar ao uses do frmPrincipal. O que faremos aqui é simples, apenas verificaremos a existência da classe TfrmAuxiliar, caso ela seja encontrada sua criação só será permitida se o CheckBox estiver marcado. Implemente o código da **Listagem 1** no evento OnClick do botão.

Listagem 1. Evento OnClick do botão

```
procedure TfrmPrincipal.Button1Click(Sender: TObject);
var
  Classe : TPersistentClass;
  Form : TForm;
begin
  Classe := GetClass(UpperCase(Edit1.Text));
  if Assigned(Classe) then
  begin
    if not CheckBox1.Checked then
      ShowMessage('Você não pode visualizar o formulário por falta de
                   permissões!')
    else
    begin
      Form := TFormClass(Classe).Create(nil);
      Form.ShowModal;
      FreeAndNil(Form);
    end;
  end
end;
```

Perceba que criamos duas variáveis, uma de tipo TPersistentClass e uma do tipo TForm. A seguir chamamos o método GetClass passando como parâmetro o conteúdo do Edit. Caso a classe digitada seja encontrada, automaticamente

teremos uma instância desta classe retornada e atribuída à variável Classe. Verificamos assim através do método assigned se realmente existe conteúdo na variável, o CheckBox não estando marcado o usuário receberá uma mensagem de que não possui privilégios, caso contrário realizamos um TypeCast da variável Classe para um TFormClass e atribuímos este resultado à variável Form, apenas para conseguirmos chamar o construtor padrão da classe e disponibilizar seus métodos. Assim o formulário é criado, exibido e logo destruído.

Se você executou sua aplicação perceberá que nada ocorreu. Isso se deve porque o nosso método GetClass verifica se a classe a qual nós estamos procurando está registrada, neste caso não possuímos nenhuma classe registrada, então nunca obteremos uma instância da mesma. O que devemos fazer neste caso é realizar o registro da classe usando o método RegisterClass. Pois bem, abra o seu frmAuxiliar e faça no final da unit a seguinte codificação:

```
initialization
 RegisterClass(TfrmAuxiliar);
end.
```

Com o RegisterClass, no momento da criação de nossa aplicação será gerado um registro de todas as classes que possuem esta instrução, fazendo assim com que seja possível através do método GetClass obter uma instância da classe digitada. Execute novamente sua aplicação e observe o resultado conforme ilustra a **Figura 2**.

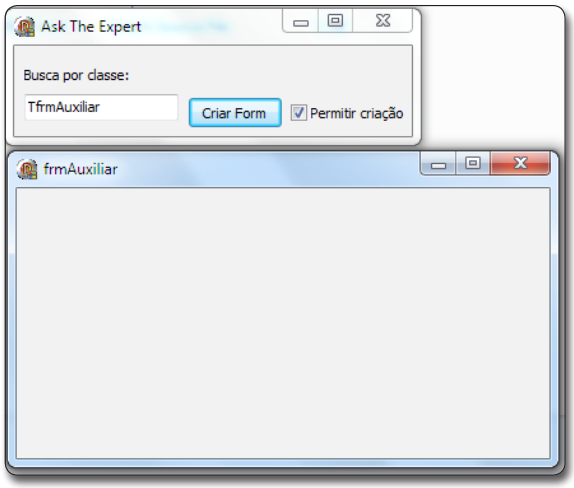


Figura 2. Execução da aplicação

Envie suas dúvidas para nossa equipe de especialistas:
guinther@devmedia.com.br
pauloquicoli@devmedia.com.br
daniel.sobrinho@devmedia.com.br
rodrigo.carreiro@devmedia.com.br

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!
Dê seu voto sobre este artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Existem coisas que não conseguimos ficar sem!

...só pra lembrar, sua assinatura pode estar acabando!

Renove Já!

www.devmedia.com.br/renovacao

Para mais informações:
www.devmedia.com.br/central

DevMedia GROUP

Quick Update:
Trabalhando com herança de Data Modules

Daniel Sobrinho Laporte
daniel.sobrinho@devmedia.com.br
daniel.laporte@gmail.com
www.twitter.com/daniel_laporte
Colunista e editor técnico da Revista ClubeDelphi. Graduado em Sistemas de informação e pós-graduado em Administração de Bancos de Dados pela Universidade de Franca. É analista de sistemas e desenvolvedor Delphi desde 2005 desenvolvendo aplicações cliente/servidor e multicamadas.

Conforme vamos criando nossos projetos é natural procurarmos maneiras de tornar nosso desenvolvimento mais produtivo, adotando práticas de programação que nos auxiliem nessas tarefas. Em Delphi e em outras linguagens temos a possibilidade de utilizar o conceito de herança para "quebrar galhos", facilitando a manutenção, centralizando a codificação e evitando o desgaste de bolar novas interfaces visuais, por exemplo, mas não é comum desenvolvermos herança em Data Modules. Talvez você possa pensar que não existem tantas justificativas para fazer isto, mas a verdade é que utilizando este recurso e um pouco de criatividade podemos desenvolver soluções muito interessantes que nos poupem tempo de projeto. Vamos montar um simples exemplo e ver na prática como isso pode ser possível. Considere a estrutura de um banco de dados conforme a **Figura 1**.

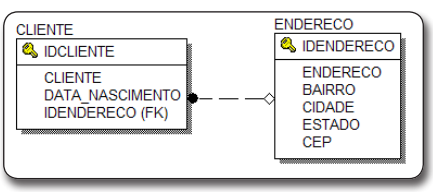


Figura 1. Modelagem do banco de dados proposto

Não entrarei em detalhes sobre como criar o banco de dados, presumindo que você já o criou conforme a estrutura da **Figura 1**. O que demonstraremos aqui são maneiras de tornar mais produtivo seu desenvolvimento, fazendo com que tarefas de configuração efetuadas por nós, sejam efetuadas de maneira automática, via codificação. Vamos supor que você deseja realizar um SELECT que obtenha as instruções de cliente e

endereço. Levando em consideração as configurações cotidianas, como ProviderFlags (nos campos) e UpdateMode nos DataSetProviders, por exemplo, gastamos certo tempo que obviamente poderia ser economizado, sem contar que toda vez que é adicionado um novo componente ao Data Module, temos que prepará-lo para receber as novas configurações. Neste caso teríamos em nosso DataSet uma instrução SQL como a da **Listagem 1**.

Listagem 1. Configuração da instrução SQL

```
SELECT
  CLIENTE.IDCLIENTE,
  CLIENTE.CLIENTE,
  CLIENTE.DATA_NASCIMENTO,
  CLIENTE.IDENDERECO,
  ENDERECO.ENDERECO AS ENDERECO_NU,
  ENDERECO.BAIRRO AS BAIRRO_NU,
  ENDERECO.CIDADE AS CIDADE_NU,
  ENDERECO.ESTADO AS ESTADO_NU,
  ENDERECO.CEP AS CEP_NU
FROM
  CLIENTE
LEFT JOIN ENDERECO ON
  CLIENTE.IDENDERECO = ENDERECO.IDENDERECO
WHERE
  CLIENTE.IDCLIENTE =:IDCLIENTE
```

Listagem 2. Classe de configuração dos métodos

```
unit untConfiguracao;

interface

uses
 Classes, Provider, DB, SqlExpr, DBClient, Dialogs, SysUtils;

 type
  TConfiguracao = class
  private
   {private methods}
  public
   class procedure AddFields(Dataset: TDataSet);
   class procedure ConfigureProviders(DataModule : TDataModule);
   class procedure ConfigureAllDataSets(DataModule : TDataModule);
  end;

implementation
...
```

Note que alteramos o nome dos campos através de um alias (apelido) para os campos. Este Alias foi definido como nome do campo acrescentado das siglas NU de "No Update", para referenciar os campos que não sofrerão atualização. Imagine agora que nossa tarefa será adicionar os campos nos DataSets, setar os ProviderFlags automaticamente e configurar os DataSetProviders. Para isso, criaremos uma classe estática que fará este trabalho para nós a partir de três métodos. Abra seu Delphi e crie uma nova VCL Application, em seguida adicione uma nova unit ao projeto, codificando-a conforme a **Listagem 2** e salvando-a como untConfiguracao.pas.

Pressione Ctrl + Shift + C para que sejam automaticamente declarados os corpos dos métodos. Se você atentar ao código da **Listagem 2,** notará que temos aqui uma classe estática como já foi comentado, portanto não há necessidade criá-la, bastando apenas sua chamada no formulário. A primeira tarefa a fazer é adicionar os campos nos DataSets. Para isso utilizamos a procedure AddFields que tem seu código descrito na **Listagem 3.**

Listagem 3. Adicionando os campos a um DataSet

```
01 class procedure TConfiguracao.AddFields(Dataset: TDataSet);
02 var
03  i : Integer;
04 begin
05  try
06   if Dataset.Active then
07    Dataset.Close;
08   if Dataset.Fields.Count > 0 then
09   begin
10    repeat
11     Dataset.Fields[DataSet.Fields.Count -
       DataSet.Fields.Count].Destroy;
12    until
13     DataSet.Fields.Count = 0;
14   end;
15   if Dataset.FieldDefs.Count > 0 then
16   begin
17    repeat
18     Dataset.FieldDefs[Dataset.FieldDefs.Count –
       Dataset.FieldDefs.Count].Destroy;
19    until
20     Dataset.FieldDefs.Count = 0;
21   end;
22   Dataset.FieldDefs.Update;
23   for i := 0 to Dataset.FieldDefs.Count - 1 do
24   begin
25    if Dataset.FindField(Dataset.FieldDefs[i].Name) = nil then
26     Dataset.FieldDefs.Items[i].CreateField(Dataset);
27   end;
28   for i := 0 to DataSet.FieldCount - 1 do
29   begin
30    if i = 0 then
31    begin
32     DataSet.Fields[0].ProviderFlags :=
       [pfInKey, pfInUpdate, pfInWhere];
33     Dataset.Fields[0].Required := False;
34    end
35    else if Copy(Dataset.Fields[i].FieldName,
36     Length(Dataset.Fields[i].FieldName) -1, 2) = 'NU' then
37     Dataset.Fields[i].ProviderFlags := []
38    else
39     DataSet.Fields[i].ProviderFlags := [pfInUpdate];
40   end;
41  except
42   on E: exception do
43    ShowMessage('Erro: ' + e.Message);
44  end;
45 end;
```

Observando a **Listagem 3** você pode ter a impressão de que o código das Linhas 08 a 21 é praticamente idêntico, porém, no primeiro destruímos todos os Fields e em seguida todos os FieldDefs. Isto se deve ao fato de que caso seja incluído algum campo à estrutura SQL, nós devemos adicioná-lo ao Fields Editor e reconfigurá-lo. O método Update faz com que os campos sejam atualizados, para que em seguida realizemos sua devida criação (Linhas 23 a 27). Por último, note que entre as Linhas 28 a 40 os campos têm seus ProviderFlags devidamente configurados. Observe que para a chave primária nós acrescentamos a informação de não requerido, para que futuramente seja possível determinar o valor diretamente pelo DataSetProvider, os campos que possuem o final 'NU' não possuirão atribuição alguma nos ProviderFlags. Caso ocorra algum erro, a mensagem será capturada e exibida conforme mostra a Linha 43.

O método ConfigureAllDataSets simplesmente lê o DataModule recebido como parâmetro, verifica se os componentes são do tipo TSQLDataSet e os passa como parâmetro para a procedure AddFields, dispensando assim a necessidade de realizar a instrução AddFields, que poderia ter sido encapsulada no escopo private caso você desejasse. Sua implementação se encontra na **Listagem 4**.

Listagem 4. Configurando os DataSets

```
class procedure TConfiguracao.ConfigureAllDataSets(
            DataModule: TDataModule);
var
 i : Integer;
begin
 for i := 0 to DataModule.ComponentCount - 1 do
 begin
  if DataModule.Components[i] is TSQLDataSet then
    AddFields(TSQLDataSet(DataModule.Components[i]));
 end;
end;
```

Finalmente o último método de nossa classe (**Listagem 5**), que também recebe um DataModule como parâmetro, realiza uma contagem de componentes (Linha 05 – 14). Encontrando um componente DataSetProvider (Linha 07) este é automaticamente configurado com o UpdateMode adequado (Linha 12) e as opções poIncFieldProps e poPropagateChanges (Linha 11).

A partir disso temos tudo pronto para realizar uma herança de DataModules. Insira um novo Data Module à sua aplicação, adicione a untConfiguracoes ao seu uses e em seguida faça as seguintes instruções no método OnCreate do mesmo:

```
TConfiguracao.ConfigureAllDataSets(Self);
TConfiguracao.ConfigureProviders(Self);
```

Listagem 5. Método de configuração dos DataSetProviders

```
01 class procedure TConfiguracao.ConfigureProviders(DataModule:
   TDataModule);
02 var
03  i : Integer;
04 begin
05  for i := 0 to DataModule.ComponentCount - 1 do
06  begin
07   if DataModule.Components[i] is TDataSetProvider then
08   begin
09    TDataSetProvider(DataModule.Components[i]).Options :=
10    TDataSetProvider(DataModule.Components[i]).Options +
11    [poIncFieldProps, poPropogateChanges];
12    TDataSetProvider(DataModule.Components[i]).UpdateMode :=
      upWhereKeyOnly;
13   end;
14  end;
15 end;
```

Note que o próprio Data Module é passado como parâmetro para as procedures. A partir disso qualquer novo DataModule descendente deste, possuirá suas configurações realizadas de forma automática. A grande vantagem da usabilidade desta classe é que você pode fazer o mesmo com suas aplicações multicamadas no modelo BSS, COM+ ou na nova arquitetura DataSnap, pois tanto os Remote Data Modules, Transactional Data Modules ou Server Modules são descendentes de TDataModule. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Faça um *up grade* em sua carreira.

Em um mercado cada vez mais focado em qualidade ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lança para os desenvolvedores brasileiros sua primeira revista digital totalmente especializada em Engenharia de Software. Todos os meses você irá encontrar artigos sobre Metodologias Ageis; Metodologias tradicionais (*document driven*); ALM (*application lifecycle management*); SOA (aplicações orientadas a servicos); Analise de sistemas; modelagem; Métricas; orientação à objetos; UML; testes e muito mais. **Assine já!**

DevMedia group

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

DataSnap 2010

Crie servidores de aplicação escaláveis utilizando REST e JSON

Rodrigo Carreiro Mourão
rodrigocarreiro@rmfactory.com.br
Analista, programador, consultor, instrutor e editor técnico da revista ClubeDelphi. Instrutor certificado pela Borland, possui ainda o título de Delphi Product Certified. É sócio da RM Factory Soluções em TI e da Szeryng Traduções e Informática. Atua como analista de TI do Grupo Editorial Nacional e é instrutor de treinamentos oficiais Embarcadero no Rio de Janeiro. Como palestrante atuou na última Borcon entre outros eventos de Delphi no Brasil. Membro ativo na comunidade e apaixonado pelo Delphi.

Do que trata o artigo
O artigo trata do desenvolvimento de servidores de aplicação utilizando a nova arquitetura DataSnap disponível no Delphi 2010. Veremos como disponibilizar serviços web aplicando os conceitos REST trabalhando com notação JSON permitindo assim que nossos serviços possam ser consumidos por aplicações desenvolvidas por praticamente qualquer linguagem de programação. No nosso exemplo utilizaremos jQuery com AJAX para consumir os serviços permitindo a você aplicar os conceitos utilizando C#, PHP, Delphi etc.

Para que serve
A arquitetura em camadas é uma das mais utilizadas em sistemas de grande porte, pois este tipo de arquitetura nos permite ter um sistema mais escalável e de fácil manutenção. Ao permitir que um servidor de aplicação possa ser consumido via web estamos aumentando consideravelmente seu nível de escalabilidade e entrando no mundo dos serviços web. Quando esta disposição dos serviços na web é feita utilizando o conceito REST estamos aplicando o que há de melhor no mercado para esta tarefa.

Em que situação o tema é útil
Desenvolver uma aplicação em camadas é útil quando queremos concentrar nossas regras de negócio em um único lugar, encapsulando a implementação, facilitando a manutenção e aumentando a escalabilidade. Publicar este serviço na web ou permitir que o mesmo seja consumido via HTTP também é extremamente útil, pois desta maneira abstraímos ainda mais os detalhes da aplicação e permitimos assim que clientes desenvolvidos em diferentes linguagens possam utilizar nossos serviços.

Resumo do DevMan

O desenvolvimento de aplicações em camadas sempre foi um capítulo à parte no Delphi. No início a tecnologia era chamada de Midas, logo depois foi batizada de DataSnap. Quando começou, a tecnologia era dependente de COM, porém com o lançamento do Delphi 2009 ela passou a funcionar sobre o framework DBExpress. Agora com o Delphi 2010 a arquitetura DataSnap deu um novo salto e desta vez bem grande. Agora podemos criar servidores DataSnap RESTful e trafegar dados com notação JSON, tudo isso nativamente no Delphi 2010. Neste artigo, após entendermos como funcionam as novas tecnologias vamos criar um servidor DataSnap, em seguida será adicionado suporte à HTTP para que na sequência nosso servidor possa ser consumido por um cliente utilizando jQuery com AJAX.

Muitos desenvolvedores fazem confusão quando o assunto é desenvolvimento em camadas. Pode soar estranho o que vou falar, mas criar uma aplicação com DataSetProvider de um lado e ClientDataSet de outro não significa que você está criando um servidor de aplicação. O termo servidor de aplicação ou aplicação em camadas vai além disso. O conceito de aplicação em camadas é poder ter em uma aplicação, geralmente chamada de servidor de aplicação, todas as suas regras de negócio e permitir que essas regras sejam consumidas por aplicações clientes. O grande equívoco é pensar que se desenvolvemos um servidor de aplicação com Delphi os clientes só podem ser aplicações Delphi, pelo contrário. O objetivo de um servidor de aplicação é justamente inverso.

É aí que muitos desenvolvedores Delphi perdiam o melhor da arquitetura DataSnap. Mesmo com a versão anterior do DataSnap era possível que uma aplicação não Delphi consumisse um servidor feito em Delphi. A mágica por trás disso chamava-se COM. Na arquitetura anterior do DataSnap para criar um servidor de aplicação bastava adicionar um RemoteDataModule. Este componente era na verdade uma coClass, uma classe com suporte à automação, em outras palavras, uma classe que pode ser instanciada remotamente. Ao adicionar um RemoteDataModule era criada uma interface onde poderíamos adicionar métodos para que fossem exportados pelo servidor. Esta interface é na verdade parte da estrutura chamada TypeLibrary, um binário que era um dos produtos da compilação do servidor de aplicação. A TypeLibrary é que era registrada na máquina e não o executável, como muitos pensavam.

Com a aplicação criada e os métodos adicionados à interface, bastava informar a quem fosse desenvolver um cliente para o nosso servidor o endereço onde o mesmo se encontrava registrado. Para facilitar o desenvolvimento poderíamos ainda enviar o arquivo com extensão .tlb para que o programador pudesse registrar localmente onde o cliente fosse rodar e então trabalhar diretamente com os métodos da interface que estava descrita na Type Library. O problema é que, na grande maioria das vezes os desenvolvedores adicionavam um RemoteDataModule e simplesmente inseriam ali alguns SqlDataSets e DataSetProviders para que pudessem, do lado cliente, conectar os ClientDataSets nestes Providers através de um componente DComConnection ou SocketConnection. Com isso o servidor de aplicação ficava restrito a clientes desenvolvidos em Delphi. É bem verdade que, da maneira como o DataSnap funcionava, não era nada fácil criar um servidor para clientes não Delphi, era trabalhoso, ainda mais se trabalhássemos com classes e objetos.

Com o passar do tempo a tecnologia COM caiu em desuso e o Delphi precisava acompanhar a evolução tecnológica. Com o lançamento do Delphi 2009 veio também a nova arquitetura, e por que não dizer, o novo framework DataSnap utilizando 100% tecnologia Delphi, uma vez que agora utiliza o DBExpress para conectar o cliente com o servidor de aplicação. Essa mudança quebrou vários paradigmas. O principal deles era a dependência da tecnologia COM, sem falar que agora poderíamos trafegar entre o cliente e o servidor todos os tipos suportados pelo DBExpress incluindo TDBXReader. Mas agora tínhamos um novo problema, se o novo DataSnap não trabalha mais com COM então só poderíamos criar clientes que possuíssem Driver DBExpress. Para a grande maioria isso não seria um problema, pois o cenário era Delphi para Delphi, ou seja, servidor de aplicação Delphi e cliente desenvolvido em Delphi, e para aqueles que desejassem desenvolver clientes Web poderiam optar pelo Delphi Prism, que possuía um Driver DBExpress e portanto poderia facilmente consumir um servidor de aplicação feito com Delphi Win32. Mas e as demais linguagens, como ficam? Imagine uma empresa que gostaria de disponibilizar alguns métodos de seu servidor para os clientes. Como fazer um pedido, consultar uma fatura e etc., se eles trabalhassem com PHP, Java, C#, C++. É exatamente neste cenário que o Delphi 2010 se encaixa.

Nota do DevMan

Component Object Model (COM) é uma plataforma da Microsoft para componentes de software lançada em 1993. Ela é usada para permitir a comunicação entre processos e a criação dinâmica de objetos em qualquer linguagem de programação que suporte a tecnologia. O termo COM é frequentemente usado no desenvolvimento de software para se referir a um grupo de tecnologias que incluem OLE, OLE Automation, ActiveX, COM+ e DCOM.

Nota do DevMan

As interfaces servem para definir apenas as assinaturas dos métodos, propriedades ou eventos a implementar nas classes. Isto permite separar a definição de uma classe da sua implementação. Em orientação a objetos uma classe que implementa uma interface deve obrigatoriamente implementar concretamente esses métodos.

Nova arquitetura DataSnap

No final de 2009 a Embarcadero anunciou o lançamento do Delphi 2010 com uma série de novidades e na área do DataSnap a maior delas. Foi adicionada à arquitetura DataSnap suporte a transporte de dados sobre HTTP, isso mesmo, agora a comunicação entre o cliente e o servidor DataSnap além de poder ser feita via TCP/IP utilizando DBExpress, pode ser feita também via HTTP sem a necessidade de nenhuma tecnologia de terceiros. A partir desta versão é possível desenvolver servidores de aplicação de forma que estes servidores possam ser consumidos por um cliente desenvolvido por praticamente qualquer linguagem de programação graças à arquitetura REST e aos dados trafegados em notação JSON. Mas o que vem a ser de fato REST e para que serve JSON?

REST

REST é a abreviação de Representational State Transfer ou Transferência de estado representacional. Ele surgiu da tese de doutorado sobre a Web escrita por Roy Fielding, que foi um dos idealizadores do protocolo HTTP. Com o advento da

Nota do DevMan

TDBXReader é uma classe do framework DBExpress 4 que fornece um leitor unidirecional para um conjunto de linhas de base de dados.TDBXReader é retornado por TDBXCommand.ExecuteQuery. O método TDBXReader.Next é utilizado para acessar a primeira linha e caso haja, as sucessivas no result set. Os valores da linha podem ser acessados usando a propriedade Value que é um array. A propriedade Value é sobrecarregada e podemos utilizar tanto a posição da coluna quanto seu nome.

Nota do DevMan

O Delphi Prism é a nova ferramenta da Embarcadero para desenvolvimento de aplicações utilizando .NET. Desde o lançamento do Delphi 2009 temos o Delphi Prism como opção para desenvolvimento Web que é na verdade uma linguagem baseada no compilador Oxygene e que roda dentro da IDE do Visual Studio.

Nota do DevMan

AJAX (acrônimo em língua inglesa de Asynchronous Javascript And XML) é o uso metodológico de tecnologias como Javascript e XML, providas por navegadores, para tornar páginas Web mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações. AJAX não é somente um novo modelo, é também uma iniciativa na construção de aplicações Web mais dinâmicas e criativas.

Nota do DevMan

Web 2.0 é a mudança para uma internet como plataforma, e um entendimento das regras para obter sucesso nesta nova plataforma. Entre outras, a regra mais importante é desenvolver aplicativos que aproveitem os efeitos de rede para se tornarem melhores quanto mais são usados pelas pessoas, aproveitando a inteligência coletiva.

Nota do DevMan

SOAP (originado do acrônimo inglês Simple Object Access Protocol) é um protocolo para troca de informações estruturadas em uma plataforma descentralizada e distribuída, utilizando tecnologias baseadas em XML. Sua especificação define um framework que provê maneiras para se construir mensagens que podem trafegar através de diversos protocolos e que foi especificado de forma a ser independente de qualquer modelo de programação ou outra implementação específica.

Nota do DevMan

XML (eXtensible Markup Language) é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais. É um subtipo de SGML (acrônimo de Standard Generalized Markup Language, ou Linguagem Padronizada de Marcação Genérica) capaz de descrever diversos tipos de dados. Seu propósito principal é a facilidade de compartilhamento de informações através da Internet. A principal característica do XML, de criar uma infraestrutura única para diversas linguagens, é que linguagens desconhecidas e de pouco uso também podem ser definidas sem maior trabalho e sem necessidade de serem submetidas aos comitês de padronização.

Web 2.0, uma série de inovações surgiu no ambiente Web. O ambiente se tornou mais colaborativo, com a participação ativa dos usuários. Novos serviços foram criados e fazem grande sucesso na internet, o Twitter é um exemplo disso. As redes sociais cresceram e se multiplicaram. O número de blogs criados também aumentou. A informação, ou melhor, o acesso a ela ficou mais democrático. Novas tecnologias como AJAX, XML, RSS, Silverlight, Flex etc. deram o ar da graça e com isso permitiram a criação de aplicações mais eficientes e iterativas. Essas mudanças obrigaram as aplicações se adaptarem a nova realidade, hoje todas as grandes redes sociais permitem integração com outros serviços na web. É possível atualizar o perfil do Facebook automaticamente ao postar algo no Twitter. Ou mesmo ser notificado quando seu autor preferido fizer uma nova postagem no Blog.

Junto com a Web 2.0 uma modalidade de aplicação também cresceu bastante, os Web Services. Os Web Services foram criados para permitir principalmente a integração máquina-máquina sem expor detalhes como plataforma, linguagens etc. Em outras palavras os Web Services vieram para permitir que aplicações desenvolvidas em diferentes linguagens, rodando sobre diferentes plataformas pudessem trocar dados entre si de forma transparente. Para que isso fosse possível era preciso estabelecer um padrão de comunicação entre as aplicações, assim foi adotado o protocolo SOAP como protocolo de comunicação. Outro detalhe era que um Web Service deveria possuir um mecanismo para descrever o que ele era capaz de fazer, foi então criado o que conhecemos hoje como WSDL, Web Service Description Language, que utiliza XML para descrever detalhes dos serviços prestados pela aplicação.

Os Web Services são até hoje uma ótima opção para disponibilização de serviços Web, porém temos algumas coisas a considerar. Primeiro, os Web Services usam protocolo SOAP sobre HTTP, ou seja, somente o HTTP não é suficiente para se trabalhar com Web Services. Segundo, os Web Services são baseados em mensagens que são transmitidas entre o cliente e o servidor, mensagens essas que são envelopadas (encapsuladas) pelo protocolo SOAP. Tudo isso fez com que usuários deste tipo de aplicação notassem uma perda do desempenho. Frente a este cenário, a utilização da arquitetura REST tem tornado mais simples e eficientes muitas aplicações que antes eram implementadas utilizando protocolo SOAP.

Alguns entendem que o REST é um novo protocolo, outros o chamam de arquitetura e alguns entendem que ele é apenas uma filosofia, uma forma de pensar, um estilo de projetar uma aplicação. Eu fico com esta última definição. O REST é uma

Nota do DevMan

Web Service Description Language (WSDL) é uma linguagem baseada em XML utilizada para descrever Web Services funcionando como um contrato do serviço. Trata-se de um documento escrito em XML que além de descrever o serviço, especifica como acessá-lo e quais as operações ou métodos disponíveis.

Nota do DevMan

CRUD é o acrônimo da expressão em língua Inglesa Create, Retrieve, Update e Delete, usada para definir quatro operações básicas usadas em bancos de dados relacionais (RDBMS) ou em interface para usuários para criação, consulta, atualização e destruição de dados.

boa prática onde o principal objetivo é construir aplicações fracamente acopladas voltadas principalmente para o ambiente Web. Ao contrário do protocolo SOAP, que é baseado em mensagens, aplicações desenvolvidas utilizando REST trabalham com o que chamamos de recursos nomeados. Por exemplo, considere a URL:

http://rodrigocarreiro.blogspot.com/2010/01/project-natal-sera-lancado-no-mes-de.html

Ela representa um recurso, um artigo de minha autoria publicado em janeiro de 2010 no meu blog e que trata de um projeto da Microsoft. Observe que na URL não há nenhum comando do tipo:

http://rodrigocarreiro.blogspot.com/getartigo?id=1234

Isso seria na verdade o envio de uma mensagem para o servidor para retornar um artigo específico. Ao digitar a URL no browser estaremos utilizando o método GET do protocolo HTTP que retornaria nosso recurso que é o artigo em questão. Em uma aplicação REST os recursos podem ser qualquer coisa, no nosso exemplo o recurso é um arquivo HTML, mas poderia ser um arquivo XML, um objeto serializado ou algo retornado em notação JSON.

O segredo das aplicações REST está em não utilizar somente o método GET do HTTP, mas também os métodos POST, PUT e DELETE. Assim podemos, por exemplo, expor o seguinte recurso:

http://www.meuservidor.com.br/clientes

Ao requisitarmos esta URL com o método GET poderíamos listar todos os clientes do sistema. Podemos fazer uma requisição ao mesmo recurso utilizando o método PUT para incluir um novo cliente, podemos utilizar o método POST para atualizar um cliente específico ou método DELETE para excluir um cliente específico. Podemos perceber assim que há uma relação entre os métodos HTTP com os métodos de CRUD que conhecemos. Esta relação pode ser vista na **Tabela 1.**

O conceito do REST está em consumir recursos via HTTP e como o Delphi 2010 adicionou ao DataSnap o transporte de dados sobre HTTP, logo fica claro que com o Delphi 2010 é possível desenvolver servidores de aplicação utilizando REST, e como sempre de maneira fácil e extremamente simples.

Método CRUD	Método HTTP
Create	PUT
Retrieve	GET
Update	POST
Delete	DELETE

Tabela 1. Mapeamento CRUD / HTTP

JSON

Conforme foi abordado no tópico sobre REST, o conceito do REST está em consumir recursos via HTTP, e em uma aplicação REST os recursos podem ser qualquer coisa, um documento HTML, TXT ou mesmo XML, porém temos que levar em consideração questões sobre performance. Quanto mais leve for o formato do recurso, mais desempenho nossa aplicação terá, e é ai que entra o JSON.

O JSON (JavaScript Object Notation) é uma espécie de padronização para transporte de informações entre ambientes heterogêneos e que tem como linguagem o Javascript. Bem, padrão, transporte, ambiente heterogêneo, tudo isso nos remete ao XML, então vem a questão: Por que JSON e não XML? O principal motivo seria pelo fato da notação JSON ser extremamente mais simples do que a XML. Não existem tags diversas nem regras complexas para escrita e muito menos mecanismos complexos de parser. Para se ter uma ideia, observe a **Figura 1**, ela mostra a sintaxe para notação JSON para um objeto.

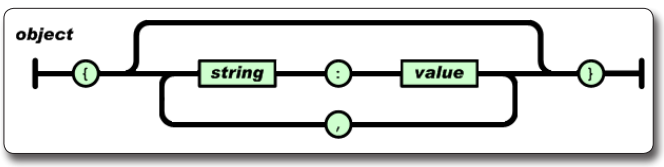


Figura 1. Notação JSON

Assim um objeto contato com nome e e-mail ficaria assim em notação JSON:

```
{"nome":"Rodrigo","email": "rodrigocarreiro@rmfactory.com.br"}
```

A simplicidade da notação JSON é tamanha que os valores em notação JSON se resumem ao que está exposto na **Figura 2.**

Atualmente, praticamente todas as linguagens de programação trabalham com JSON e sendo assim possuem componentes, classes, assembly's etc. com métodos e rotinas para facilitar a manipulação de dados em notação JSON. No Delphi 2010 não é diferente, novas units foram adicionadas assim como novas

classes para trabalhar com notação JSON. Entre as novas units podemos destacar a unit DBXJSON, que possui a declaração das principais classes para trabalhar com JSON como, por exemplo, TJSONValue, TJSONString, TJSONNumber, TJSONArray etc.

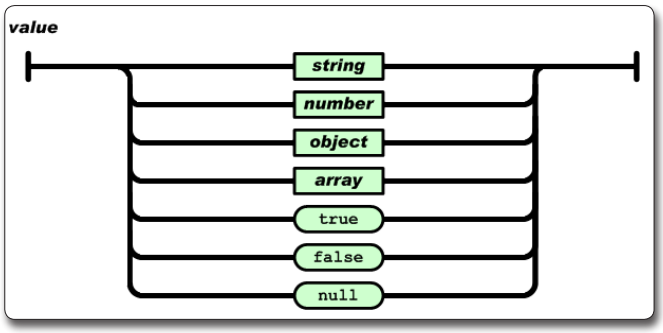


Figura 2. Tipos na notação JSON

Como já foi citado, no Delphi 2010 foi adicionado ao DataSnap suporte a transporte via HTTP e os dados trafegados por este protocolo na arquitetura DataSnap são em formato JSON. Não era para menos, trafegando os dados em notação JSON conseguimos dois grandes feitos, um excelente desempenho das aplicações devido à levesa, característica da notação e a escalabilidade uma vez que praticamente todas as linguagens de programação suportam JSON.

Depois de todo este embasamento teórico nada melhor do que colocar todo este conceito em prática, então mãos à obra.

Montando o ambiente

Para fixarmos os conceitos aprenderemos a desenvolver servidores de aplicação baseados em REST. Vamos desenvolver um exemplo onde disponibilizaremos um recurso via HTTP, que em nosso caso será uma lista de clientes cadastrados em nossa base de dados. Para facilitar vamos utilizar o banco de dados Employee.fdb que vem como banco de dados de exemplo no Firebird.

Nota

Para esta exemplo será utilizado o banco de dados Firebird 2.1, mas você pode ficar a vontade para escolher o banco de dados que melhor lhe convir. Vale lembrar que poderá ser usado qualquer outro repositório de dados, como arquivos XML por exemplo.

Com o banco definido vamos ao Delphi 2010 e criar uma nova aplicação. Vamos utilizar um Wizard para criar uma aplicação com os principais componentes necessários para desenvolver um servidor de aplicação. Vá ao menu File > New > Other e na seção DataSnap Server selecione a opção DataSnap Server. A **Figura 3** mostra a opção que devemos selecionar.

Feito isso um wizard será aberto. Neste wizard podemos configurar que tipo de aplicação será usada para criar nosso servidor, podemos criar uma aplicação VCL normal, uma aplicação Console ou então um Service Application, para este exemplo ficaremos com uma aplicação VCL normal. A segunda configuração trata sobre qual protocolo de comunicação vamos usar no nosso servidor, se TCP/IP, HTTP ou ambos. No caso de HTTP temos a opção de adicionar suporte à autenticação via HTTP, como nosso objetivo é utilizar REST vamos selecionar ambas as opções, TCP/IP e HTTP e neste exemplo não trabalharemos com autenticação. A terceira e última seção de configuração diz respeito à adição ou não de um ServerMethod e funções de exemplo. Vamos deixar as opções marcadas somente para que possamos entender como funciona uma requisição ao servidor via HTTP. A **Figura 4** mostra o wizard com as opções que devemos selecionar.

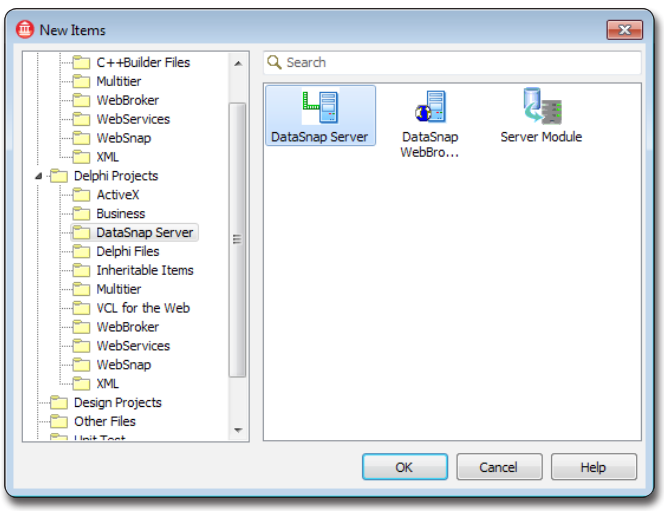


Figura 3. Criando a aplicação

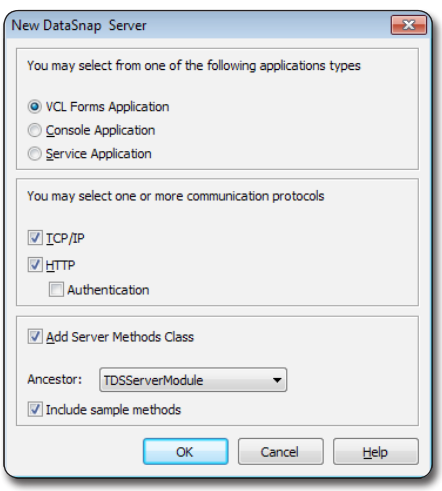


Figura 4. Wizard para criação da aplicação

Feito isso será criada uma nova aplicação com três estruturas: o formulário principal da aplicação, um TServerModule chamado ServerMethods1 e um TDataModule chamado ServerContainer1. A classe TServerModule é uma nova estrutura específica para trabalhar com o novo DataSnap. Ele na verdade descende de um TDataModule e possui duas principais alterações, a primeira é que em sua estrutura hierárquica existe uma classe chamada TProviderDataModule que possui métodos para registrar todos os DataSetProviders que por ventura

venham a ser adicionados no Server Module. A segunda alteração é que esta classe TServerModule possui as diretivas de compilação {$MethodInfo ON} e {$MethodInfo OFF}. Isso é necessário para que os métodos que venham ser adicionados ao Server Module possam ser lidos pelo mecanismo de reflexão do Delphi e assim possam, entre outras coisas, ser acessados pelos clientes.

Repare que neste ServerMothods1 foi adicionada uma função de exemplo pois no momento de criação da aplicação deixamos esta opção selecionada. Como esta função está dentro de um Server Module e o mesmo possui em sua estrutura hierárquica as diretivas de compilação, então este método poderá ser exportado para os clientes. Veremos isso mais à frente. Agora todo o servidor de aplicação está na verdade no ServerContainer1, nele temos a presença de quatro componentes que foram adicionados pelo wizard. Na **Tabela 2** temos as informações de nome, classe, função, incidência por aplicação e observação.

Todas as conexões entre os componentes já foram realizadas pelo wizard e uma rápida olhada neles lhe dará uma visão de como eles interagem entre si. A única consideração que é importante fazer é sobre o evento onGetClass do componente TDSServerClass, repare que ele já veio com implementação. Este evento é disparado sempre que um método da classe que o ServerClass expõe for invocado. Quando o método for invocado temos que enviar, via parâmetro por referência, a classe que está sendo exposta por este ServerClass. A implementação é bem simples, apenas passamos para o parâmetro PersistentClass a classe a ser exposta, neste caso ServerMethodsUnit1.TServerMethods1.

Resumindo, o wizard criou três estruturas, uma delas um Server Module que é exposto pelo componente DSServerClass1 através do evento OnGetClass. Agora basta executar a aplicação para que o servidor entre em operação, sem a necessidade de qualquer registro ou configuração, no máximo liberar a porta no firewall do Windows, pois o componente DSHTTPService1 utiliza por padrão a porta 80 para que possamos utilizar o contexto REST. Com a aplicação em execução podemos abrir um navegador e fazer a requisição ao servidor de aplicação utilizando o conceito REST.

Nota

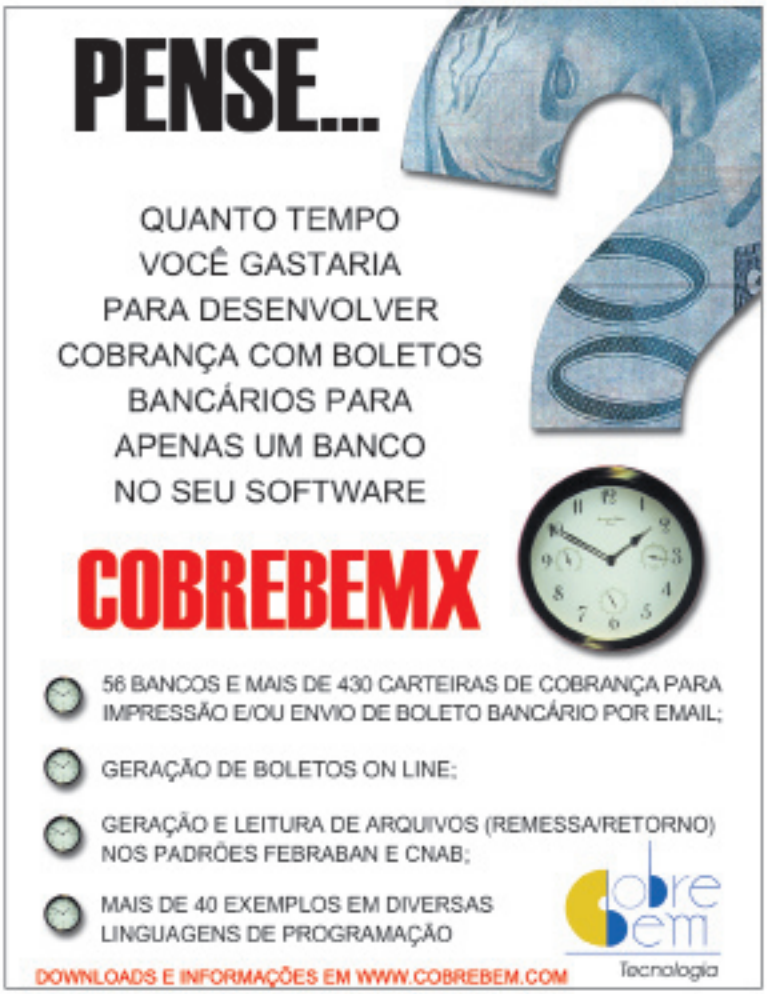
É importante citar que a porta 80 é utilizada por diversos programas, principalmente os voltados para web, como por exemplo o IIS, Apache, etc. Caso a porta 80 esteja sendo usada por outra aplicação o servidor irá gerar uma exceção quando for executado. Se isso ocorrer basta alterar a porta usada pelo servidor de aplicação através da propriedade HttpPort do componente DSHttpService1.

Nota do DevMan

As diretivas de compilação são comandos que não são compilados, sendo dirigidos ao pré-processador, que é executado pelo compilador antes da execução do processo de compilação propriamente dito. Portanto, o pré-processador modifica o programa fonte, entregando para o compilador um programa modificado. Todas as diretivas de compilação no Delphi são declaradas dentro de um par de chaves precedidas pelo caractere {$DIRETIVA}. As diretivas podem ser colocadas em qualquer parte do programa. {$IFDEF VER150} é um exemplo de diretiva que verifica a versão do compilador do Delphi onde a aplicação está sendo compilada.

Nota do DevMan

Reflexão computacional ou somente reflexão é a capacidade de um programa observar ou até mesmo modificar sua estrutura ou comportamento. Quando o código fonte de um programa é compilado, a informação sobre sua estrutura normalmente se perde quando o código de baixo nível é produzido. Já num sistema que suporta reflexão, essa informação é preservada como metadados, anexados ao código gerado. No Delphi a reflexão é implementada através de um recurso chamado RTTI, Runtime Type Information.



Nome	Classe	Função	Incidência
DSServer1	TDSServer	Componente que transforma uma aplicação em um servidor de aplicação.	Um por aplicação
DSTCPServerTransport1	TDSTCPServerTransport	Componente de transporte de dados entre o cliente e o servidor via protocolo TCP/IP.	Um por aplicação
DSHTTPService1	TDSHTTPService	Componente de transporte de dados entre o cliente e o servidor via protocolo HTTP.	Um por aplicação
DSServerClass1	TDSServerClass	Componente que expõe uma classe para que a mesma possa ter seus métodos consumidos pela aplicação clientes. Somente classes que descendem de TPersistent e possuam as diretivas {$MethodInfo ON} e {$MethodInfo OFF} em sua estrutura.	Um por classe a ser exportada

Tabela 2. Descrição dos componentes

Para acessar o servidor via HTTP precisamos utilizar uma URL que segue uma sintaxe básica:

http://endereco-do-servidor/datasnap/rest/Classe/Metodo/Parametro1/Parametro2/ParametroN.

Vamos entender cada parte desta URL. Em endereço-do-servidor você deverá informar o ip ou o host de onde o servidor de aplicação estará rodando. Atente para o fato de incluir a porta no endereço caso utiliza outra porta que não a 80, assim o endereço ficaria endereço-do-servidor:porta. Após o endereço do servidor temos a palavra DataSnap que faz parte da estrutura da requisição ao servidor. Na sequência temos o contexto rest que por default é justamente a palavra rest, mas isso pode ser mudado alterando a propriedade RESTContext para a palavra que desejar.

Até aqui a URL está referenciando o contexto rest de um servidor de aplicação qualquer, porém um servidor pode expor várias classes e cada uma destas, vários métodos. Por isso na URL devemos informar a classe que queremos acessar, o método desta classe que queremos invocar e caso haja, um ou mais parâmetros sempre na ordem em que foram declarados. Visto isso, para invocar nosso método echostring (método de exemplo criado) da classe TServerMethods1 devemos fazer requisição à seguinte URL HTTP://localhost/datasnap/rest/TServerMethods1/EchoString/DevMedia_Group. A **Figura 5** exibe a resposta do servidor de aplicação no browser.

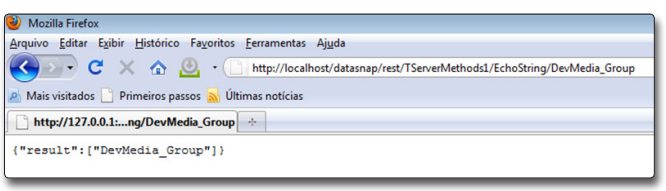


Figura 5. Resposta do servidor de aplicação

Repare que a resposta do servidor é em formato JSON, que já foi exposto no início do artigo. O mais importante é que sem empregar muito esforço estamos aplicando o conceito de REST em nosso servidor, observe. Estamos expondo um recurso, que neste caso é a classe TServerMethods1, fizemos a requisição utilizando o método GET do HTTP e recebemos este recurso em formato JSON. Tudo como foi explicado acima. Óbvio que este não é o nosso exemplo final, foi apenas uma rápida explicação de como funciona na prática a construção de um servidor de aplicação com suporte a HTTP.

Iniciando o exemplo

Para desenvolver nosso exemplo vamos aproveitar a aplicação que montamos quando criamos o ambiente e fazer apenas alguns ajustes. Primeiro vamos remover da aplicação nosso DSServeModule ServerMethods1. Isso porque não iremos expor métodos adicionados a um Server Module e sim criar uma classe própria nossa para fazer isso.

Uma vez tendo removido o item em questão, exclua a implementação do evento OnGetClass do componente DSServerClass. O próximo passo é renomear os componentes no ServerContainer1 assim como o próprio Server Container e o formulário principal da aplicação. Como sugestão podemos alterar os nomes dos componentes DSServer1, DSTCPServerTransport1, DSHTTPService1 e DSServerClass1 para DSServerDevMedia, DSTCPDevMedia, DSHTTPDevMedia e SCContato respectivamente. O formulário principal altere para FrmPrincipal e o ServerContainer para SCPrincipal.

A ideia do nosso exemplo é disponibilizar a lista de clientes cadastrados em nossa base de dados, para isso vamos criar uma classe TClientes e fazer com que ela se torne o recurso que será publicado para as aplicações clientes através do conceito de REST. Sendo assim vamos adicionar uma nova unit ao projeto e salvá-la com o nome de uClientes.pas. Nesta unit será criada a classe que estaremos expondo através do nosso servidor REST. O código da **Listagem 1** exibe a declaração completa da nossa classe TCliente.

Nota

A classe TCliente criada na **Listagem 1** não possui propriedades, nem métodos GET ou SET. Isso se deu devido o objetivo do artigo ser REST e não OO. Em uma aplicação real poderíamos ter uma classe TCliente, que seria a classe padrão de negócio, e a classe TClienteREST que seria utilizada para Expor os métodos via REST.

Para atender aos requisitos da nova arquitetura DataSnap nossa classe contato herda de TPersistent e possui as diretivas de compilação para permitir que seus métodos sejam mapeados por reflexão. Na classe contato temos dois métodos, um que retorna os dados em notação JSON e outro que retorna os dados através da classe TDBXReader. O primeiro será utilizado no contexto REST já o segundo será utilizado por clientes Win32 desenvolvidos em Delphi.

A **Listagem 1**, apesar da quantidade de código é muito simples. Nas linhas 14 e 15 nós declaramos duas variáveis dos tipos novos do DBXExpress 4, uma para conexão com o banco de dados e outra para executar comandos SQL no banco de dados. Já nas

Listagem 1. Classe TCliente

```
1.  unit uCliente;
2.  interface

3.  uses
4.    Classes, DBXCommon, DBXJSON, DbxFirebird, SysUtils;

5.  type
6.    {$METHODINFO ON}
7.    TCliente = class(TPersistent)
8.    public
9.      function GetCustomersJson(OrderBy: string): TJSONArray;
10.     function GetCustomersDBX(OrderBy: string): TDBXReader;
11.   end;
12.   {$METHODINFO OFF}
13.   var
14.     Con: TDBXConnection;
15.     DataSet: TDBXCommand;

16.   implementation

17.   function TCliente.GetCustomersDBX(OrderBy: string): TDBXReader;
18.   var
19.     Sql: TStringBuilder;
20.   begin
21.     Sql := TStringBuilder.Create;
22.     Sql.Append('Select Cust_no, Customer, Address_Line1, ');
23.     Sql.Append('      City, Country, Phone_no From Customer ');
24.     if OrderBy <> EmptyStr then Sql.Append(' Order by '+OrderBy);
25.     DataSet.Text := Sql.ToString;
26.     Result := DataSet.ExecuteQuery;
27.   end;

28.   function TCliente.GetCustomersJson(OrderBy: string): TJSONArray;
29.   var
30.     Sql: TStringBuilder;
31.     Reader: TDBXReader;
32.     Obj: TJSONObject;
33.     I: Integer;
34.   begin
35.     Result := TJSONArray.Create;
36.     Sql := TStringBuilder.Create;
37.     Sql.Append('Select Cust_no, Customer, Address_Line1, ');
38.     Sql.Append('      City, Country, Phone_no From Customer ');
39.     if OrderBy <> EmptyStr then Sql.Append(' Order by '+OrderBy);
40.     DataSet.Text := Sql.ToString;
41.     Reader := DataSet.ExecuteQuery;
42.     Obj := TJSONObject.Create;
43.     for I := 0 to Reader.ColumnCount - 1 do
44.     begin
45.       Obj.AddPair(Reader.ValueType[I].Name,TJSONString.Create
          (Reader.ValueType[I].Name));
46.     end;
47.     Result.AddElement(Obj);
48.     while Reader.Next do
49.     begin
50.     Obj := TJSONObject.Create;
51.     Obj.AddPair('Codigo', TJSONNumber.Create(Reader.Value[0].GetInt32));
52.     Obj.AddPair('Nome', TJSONString.Create(Reader.Value[1].GetAnsiString));
53.     Obj.AddPair('Endereco', TJSONString.Create(Reader.Value[2].GetAnsiString));
54.     Obj.AddPair('Cidade', TJSONString.Create(Reader.Value[3].GetAnsiString));
55.     Obj.AddPair('Pais', TJSONString.Create(Reader.Value[4].GetAnsiString));
56.     Obj.AddPair('Telefone', TJSONString.Create(Reader.Value[5].GetAnsiString));
57.       Result.AddElement(Obj);
58.     end;
59.   end;

60.   initialization
61.     Con := TDBXConnectionFactory.GetConnectionFactory.GetConnection
          ('DBDataSnap','SYSDBA', 'masterkey');
62.     DataSet := Con.CreateCommand;

63.   end.
```

linhas 61 a 62 criamos uma seção initialization para que tanto o objeto TDBXConnection e TDBXCommand sejam criados para serem consumidos. Repare que na linha 61 estamos passando um parâmetro DBDataSnap que na verdade é o nome registrado da conexão no Data Explorer. Em seguida temos a implementação dos métodos declarados nas linhas 9 e 10.

Nas linhas 17 a 27 temos método GetCustomersDBX, que tem por objetivo retornar os dados dos clientes cadastrados na base de dados utilizando a classe TDBXReader. Na linha 19 temos uma variável do tipo TStringBuilder, instanciamos o objeto na linha 21 e nas linhas 22 e 23 informamos a instrução SQL para retornar os dados do banco. Na linha 24 nós testamos o parâmetro OrderBy e caso algo tenha sido informado incluímos no comando SQL uma cláusula OrderBy. Na linha 25 passamos o comando SQL devidamente montado para o objeto DataSet e em seguida executamos o comando e retornados no result set, que é um objeto TDBXReader, através do result da função. Isso é o suficiente para que um cliente desenvolvido com Delphi win32 possa exibir os dados dos clientes através deste método.

Nas linhas 28 a 59 temos a implementação do método GetCustomersJson, cujo objetivo é o mesmo que o método anterior porém o formato de retorno será JSON, para que assim aplicações clientes desenvolvidas em outras lingugens possam consumir nosso servidor de aplicação. Nas linhas 30 a 33 temos as variáveis que utilizaremos neste método. Além de um Objeto TStringBuilder vamos precisar de um objeto TDBXReader, para receber o result set e um objeto do tipo TJsonObjeto, que será instanciado a cada linha retornada pelo select para que posteriormente possa ser adicionado no array de objetos Json, TJsonArray, que é instanciado na linha 35. Os códigos das linhas 36 a 40 são idênticos ao do método anterior e dispensam comentários. Na linha 41 executamos nossa consulta e o resultset é colocado na variável Reader.

Em seguida criamos um Objeto JSON, pois nas linhas 43 a 46 nós percorremos todas as colunas retornadas no select e para cada coluna encontrada adicionamos ao objeto JSON um par de nome=valor, ou seja, adicionamos ao objeto atributos que possuem um nome, no caso informamos o nome da coluna, e um valor que tembém passamos o nome da coluna em questão. Com isso a primeira linha do nosso array conterá os nomes das colunas retornadas no nosso select, e esse primeiro objeto é colocado dentro no array na linha 47.

Nas linhas 48 a 58 iniciamos um loop e para cada linha retornada criamos um JSONObject (linha 50), adicionamos os atributos a ele informando o nome do atributo estaticamente e o valor vindo do objeto Reader conforme método anterior (linhas 51 a 56) e por fim o adicionamos no array (linha 57). Com isso finalizamos nossa classe cliente e junto com ela nosso servidor de aplicação. Caso queira testar, execute a aplicação, abra seu browser, e informe a URL para retornar os clientes em formato JSON.

Aplicação Cliente – Win32

Vamos testar nossa aplicação criando um cliente datasnap bem simples. Para isso crie um nova aplicação no Delphi 2010 e a ela adicione um ClientDataSet, um DataSource e um DBGrid. Após isso vamos criar nossa conexão com o servidor de aplicação, lembre-se de manter o servidor em execução caso contrário não teremos êxito na conexão. Para criarmos a conexão é muito simples, vá ao DataExplorer e crie uma nova conexão utilizando o driver DataSnap. Com a conexão criada vamos modificá-la e passar os parâmetros necessários para conectar ao nosso servidor. Em protocol informe tcp/ip, em host você deve informar o endereço do servidor, que em nosso caso é localhost ou 127.0.0.1 e por fim a porta padrão, que é 211. Feche o diálogo de configuração e arraste item de conexão do DataExplorer para o formulário principal da aplicação. Isso fará com que um componente SqlConnection seja adicionado ao formulário devidamente conectado. A **Figura 6** exibe o nosso formulário.

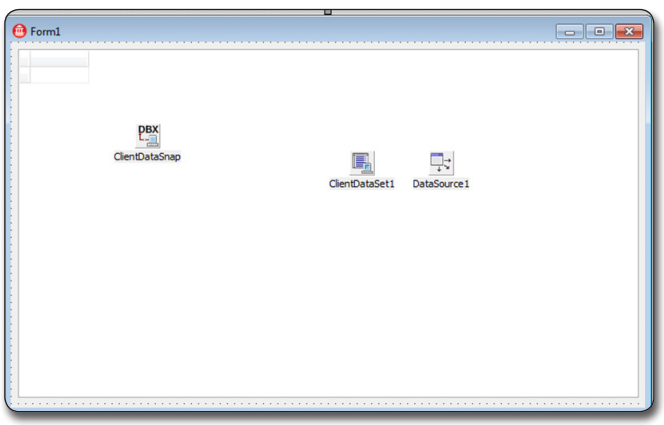


Figura 6. Formulário da aplicação cliente

Conecte os componentes de tela de acordo com o padrão: DBGrid ao DataSource e DataSource ao ClientDataSet. Agora vem a parte mais importante, clique com o botão direto sobre o SqlConnection e clique na opção Generate DataSnap Client Classes. Ao fazer isso o Delphi irá se conectar ao servidor de aplicação e para cada DSServerClass adicionado no servidor que exporta uma classe ele irá gerar uma unit do lado do cliente e nesta unit será criada o que chamamos de classe Proxy, ou classe Client.

Esta classe é um clone da classe que está no servidor, porém ao invés da implementação dos métodos temos na verdade o código necessário para que o método seja chamado no servidor através do framework DBExpress. Com isso podemos no cliente instanciar um objeto e quando invocarmos um método deste objeto esta chamada será repassada para o método da classe que está no servidor facilitando assim o consumo dos métodos no servidor de aplicação.

Salve a unit criada com o nome de uClienteProxy.pas e no formulário principal da aplicação adicione à seção uses esta nova unit. Tendo feito isso vá ao evento OnCreate do formulário e insira o código da **Listagem 2** e no evento OnTitleClick do DBGrid adicione o código da **Listagem 3.**

O códigos da **Listagem 2** e **3** são bem simples. Temos em ambos uma variável do tipo TClienteClient que é a classe gerada automaticamente pelo SqlConnection. Começamos testando a conexão com o servidor, caso não estejamos conectados realizamos a conexão. Em seguida nós instanciamos um objeto do tipo da classe Proxy (TClienteClient) e repare que o construtor desta classe requer um TDBXConnection. Isso se faz necessário para que a classe saiba como chegar no servidor, em nosso exemplo nós utilizamos o próprio SqlConnection e passamos a propriedade DBXConnection.

Listagem 2. Código do evento OnCreate

```
var
  Cliente: TClienteClient;
begin
  if not ClientDataSnap.Connected then
       ClientDataSnap.Connected := True;
  Cliente := TClienteClient.Create(ClientDataSnap.DBXConnection);
  TDBXDataSetReader.CopyReaderToClientDataSet(
       Cliente.GetCustomersDBX(''),ClientDataSet1);
  ClientDataSet1.Open;
  FreeAndNil(Cliente);
end;
```

Listagem 3. Código do evento OnTitleClick

```
var
  Cliente: TClienteClient;
begin
  if not ClientDataSnap.Connected then
       ClientDataSnap.Connected := True;
  Cliente := TClienteClient.Create(ClientDataSnap.DBXConnection);
  TDBXDataSetReader.CopyReaderToClientDataSet(
       Cliente.GetCustomersDBX(Column.Title.Caption),ClientDataSet1);
  ClientDataSet1.Open;
  FreeAndNil(Cliente);
```

Na sequência nós utilizamos uma das classes utilitárias do DBXExpress chamada TDBXDataSetReader. Esta classe possui dois métodos principais: um retorna um objeto TClientDataSet baseado em um DBXReader e o outro, que estamos usando aqui, pega um objeto DBXReader e passa seu conteúdo para um ClientDataSet, criando os Fields e adicionando seus valores.

É neste momento que encontramos a única diferença nos códigos dos dois eventos, no onCreate nós chamados o método GetCustomersDBX e informamos um conteúdo vazio para o parâmetro orderby. Já no evento onTitleClick nós

Nota

Para utilizar a classe TDBXDataSetReader e outras classes utilitárias para trabalhar com objetos TDBXReader você precisa dar uses em DBXDBReaders.

Nota

O código do evento onTitleClick é apenas didático. Não justifica fazer uma nova requisição ao servidor apenas para reordenar um resultset. Este mesmo comportamento pode ser obtido inserindo o código ClientDataSet1.IndexFieldNames := Column.FieldName; no evento onTitleClick do DBGrid.

passamos o título da coluna através do parâmetro Column (Column.Title.Caption), que vem a ser o nome do campo no banco da dados. Desta forma ao clicarmos no título da coluna faremos uma nova requisição ao servidor e receberemos um novo DBXReader com os dados ordenados pela coluna clicada. Feito isso, execute a aplicação e teste o exemplo. A **Figura 7** exibe a aplicação cliente Win32 em execução.

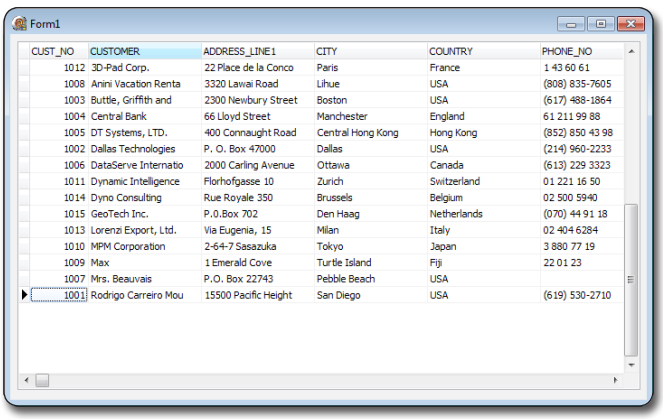


Figura 7. Aplicação cliente em execução

jQuery

Antes de criarmos nossa aplicação cliente web vamos entender o que vem a ser jQuery. Essa é uma poderosa biblioteca JavaScript criada para simplificar a criação de efeitos visuais e de interatividade em web sites. jQuery nos permite a criação de scripts de uma forma tão simples e intuitiva que consegue com algumas linhas os mesmos efeitos de um script de várias linhas desenvolvido com JavaScript tradicional. Simplicidade foi a palavra chave para John Resig na criação da biblioteca. O lema inclusive que está no site oficial é "Escreva menos, faça mais".

Com jQuery você pode facilmente manipular elementos HTML do documento adicionando eventos, alterando propriedades, aplicando estilos e tudo isso escrevendo o mínimo possível. A jQuery tem como base a função $(). É ela que gerencia tudo o que você quiser implementar. Para manipular atributos do documento HTML poderíamos fazer $(document). Se quiser adicionar um evento a todos os links do documento poderia fazer conforme o código a seguir:

```
$("a").click(function(){
  alert("Link Clicado!");
});
```

O mais interessante é que existem dezenas e dezenas de plugins para jQuery que vão desde plugins para efeitos visuais até plugins para trabalhar de forma mais eficiente com AJAX. Há também na internet uma série de APIs baseadas em jQuery disponíveis para serem utilizadas. Na seção links no final do artigo você encontra algumas delas. O assunto jQuery é muito, mas muito extenso e necessita de um artigo à parte para podermos nos aprofundar, mas para o que vamos fazer no nosso artigo essa rápida explicação é o suficiente.

Aplicação Cliente - Web

Nossa aplicação cliente na verdade será uma página HTML simples com funções javascript implementadas com jQuery. Eu optei por este tipo de cliente, pois fica fácil aplicar os conceitos mostrados aqui em aplicações ASP.Net, PHP, Java etc., uma vez que todas as linguagens para Web suportam JavaScript. Ainda no nosso projeto cliente criado no Delphi 2010 acesse o menu File > New > Other e na seção Web Documents adicione um arquivo HTML. Salve-o como clientes.html. Com o documento criado adicione ao mesmo o código da **Listagem 4.**

Listagem 4. Cabeçalho do arquivo contato.html

```
<html>
<head>
<title>Clube Delphi 119 - Cliente DataSnap com jQuery e AJAX
</title>
    <script src="http://ajax.microsoft.com/ajax/jQuery/jquery-1.4.2.js" type=
      "text/javascript"></script>
 <script src="script/clientdatasnap.js" type="text/javascript"> </script>
</head>
<body>
    <div id="header" align="center" style="font-size=30">
      Cliente DataSnap com jQuery e AJAX</div>
    <p>Lista de Clientes
    <hr size="1" width="100%">
    <div id="dvclientes"></div>
    <hr size="1" width="100%">
    <div id="footer" align="center">Exemplo desenvolvido para a revista
    CD119 - DevMedia Group
    </div>
</body>
</html>
```

Na **Listagem 4** temos o código do nosso arquivo HTML e temos que fazer apenas duas considerações:

1. Na seção head do HTML estamos importando dois arquivos javascript: o primeiro é o principal deles que é na verdade o arquivo contendo todo o código para se trabalhar com jQuery. Eu optei por trabalhar com o arquivo disponível no repositório da Microsoft. O segundo arquivo javascript (clientdatasnap.js) será criado na sequência e conterá o código necessário para que possamos consumir os métodos no servidor de aplicação;

Nota

Neste artigo optamos por utilizar a versão 1.4.2 da jQuery disponível no site da Microsoft. Há outras API baseadas em jQuery disponíveis na Web, na seção link no final do artigo você encontra algumas delas. Você ainda tem a opção de baixar o arquivo javascript para sua máquina para trabalhar localmente.

Nota do DevMan

O elemento div foi criado com o HTML 4 com a finalidade de fornecer um mecanismo genérico para agrupar e prover estrutura aos documentos. O elemento div é um container nível de bloco que possui uma id e/ou class que fornece uma identidade a uma div. Um elemento div é simplesmente um elemento container.

2. Foi adicionada no corpo do documento uma DIV chamada dvclientes. É nesta DIV que serão adicionados os dados vindos do servidor de aplicação.

Disponha os controles como achar melhor, a **Figura 8** mostra uma sugestão de interface.

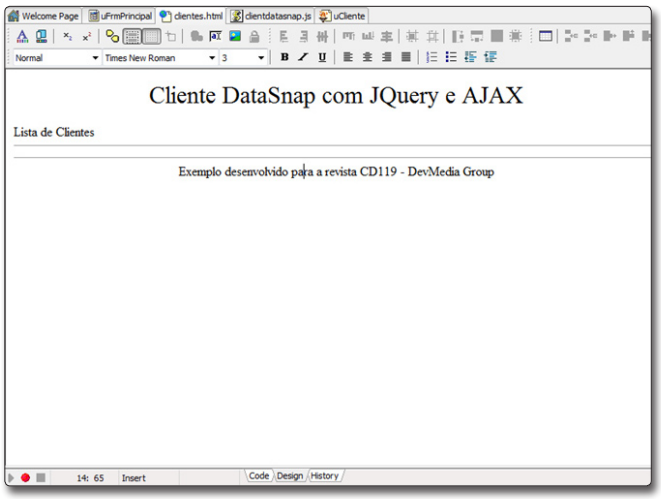


Figura 8. Interface HTML

HTML criado vamos agora dar início à codificação javascript para interagir com o nosso servidor de aplicação. Para deixar nosso exemplo um pouco mais organizado vamos colocar os códigos javascript em um arquivo exclusivo, faça isso através do File > New > Other e na seção Web Documents selecione a opção JavaScript. Salve os arquivos como clientdatasnap.js de preferência em uma pasta script na raiz do seu projeto. Feito isso adicione a este arquivo o código da **Listagem 5.**

Na linha 1 iniciamos nosso código criando uma variável com o conteúdo da URL base para todas as requisições que iremos fazer ao servidor facilitando assim a escrita dos demais códigos. Os códigos das linhas 2 a 27 declaram a função getclientes que tem por objetivo retornar do servidor a lista de contatos cadastrados na base de dados. Entre as linhas 3 e 7 testamos o parâmetro order para saber se vamos chamar o método ordenando por um campo ou não. Com base neste teste montamos nossa uri para que em seguida possamos fazer a requisição ao servidor.

Feito isso utilizamos o método getJSON (linha 8) que recebe dois parâmetros: o primeiro é a URL utilizada que será invocada via método GET ao servidor e o segundo parâmetro é uma função de callback, ou seja, a função que seja executada caso a requisição seja feita com sucesso (linhas 9 a 28).

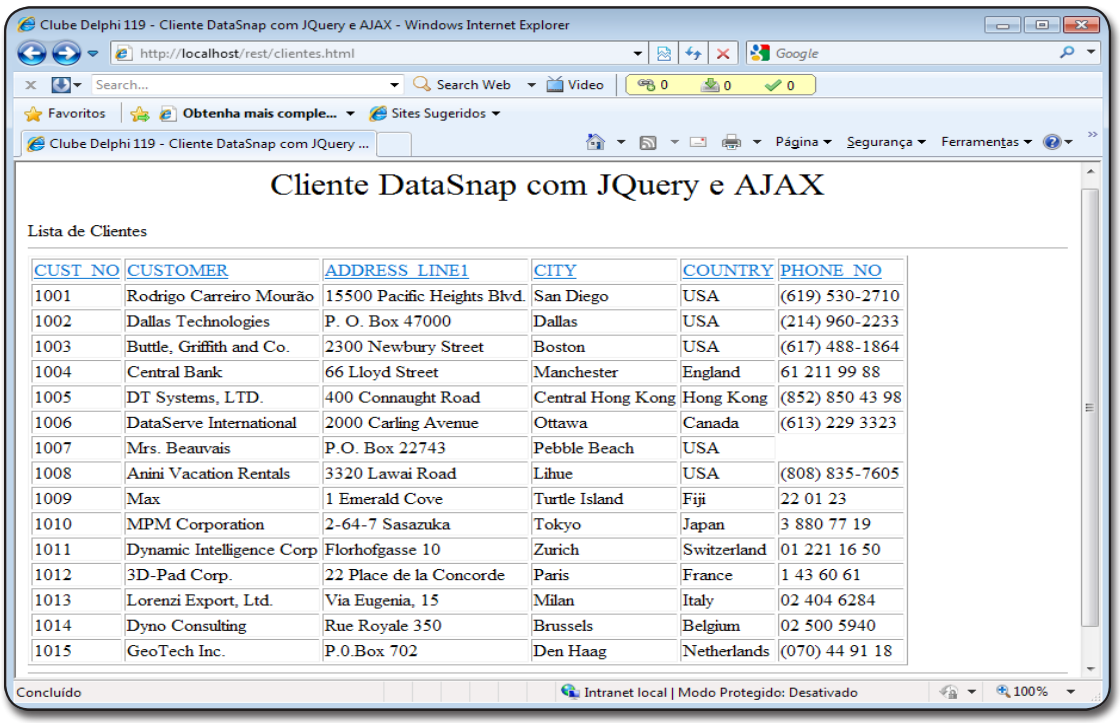
O parâmetro data da função de callback representa o retorno do servidor, que em nosso caso é um array em notação JSON. Todo retorno do DataSnap no Delphi é feito em um array chamado result. Sendo assim, na linha 10, passamos para uma variável clientes o primeiro item do array result representado aqui pelo parâmetro data, esse item nada mais é do que a lista de contatos em formato JSON. Na linha 11 criamos outra variável onde iremos concatenar tags HTML para montar uma tabela que irá exibir os dados dos contatos vindos do servidor.

Listagem 5. Código do arquivo clientdatasnap.js

```
1.  var baseUrl = "http://localhost:90/dataSnap/rest/TCliente/";
2.  function getclientes(order){
3.  if (typeof order == "undefined"){
4.  var uri = "GetCustomersJson";
5.  }else{
6.  var uri = "GetCustomersJson/"+order;
7.  }
8.  $.getJSON(baseUrl+uri,
9.  function(data) {
10. clientes = data.result[0];
11. var grid = "<table border='1'>";
12. for (var i=0; i < clientes.length; i++){
13. if ( i == 0 ){
14. grid = grid + "<tr> <td>"+"<a href='#'>"+clientes[i].CUST_NO+"</a></td>"+
15. "<td>"+"<a href='#'>"+clientes[i].CUSTOMER+"</a></td>"+
16. "<td>"+"<a href='#'>"+clientes[i].ADDRESS_LINE1+"</a></td>"+
17. "<td>"+"<a href='#'>"+clientes[i].CITY+"</a></td>"+
18. "<td>"+"<a href='#'>"+clientes[i].COUNTRY+"</a></td>"+
19. "<td>"+"<a href='#'>"+clientes[i].PHONE_NO+"</a></td></tr>";
20. }else{
21. grid = grid + "<tr> <td>"+clientes[i].Codigo+"</td>"+
22. "<td>"+clientes[i].Nome+"</td>"+
23. "<td>"+clientes[i].Endereco+"</td>"+
24. "<td>"+clientes[i].Cidade+"</td>"+
25. "<td>"+clientes[i].Pais+"</td>"+
26. "<td>"+clientes[i].Telefone+"</td></tr>";
27. }
28. }
29. grid = grid + "</table>";
30. $("#dvclientes").html(grid);
31. $("#dvclientes").find("a").click(
32. function(e){
33. var link = $(this);
34. getclientes(link.html());
35. });
36. });
37. }
38. $(document).ready(function() {
39. getclientes();
40. });
```

Das linhas 12 a 27 executamos uma loop e a cada item do array retornado concatenamos na variável grid, uma tag <TR> e duas tags <TD> e dentro destas tags <TD> adicionamos o conteúdo de cada item do array clientes. Como a primeira linha do nosso array é na verdade o nome das colunas, então na linha 13 verificamos se estamos mapeando a primeira linha do nosso array e em caso afirmativo damos um tratameno diferenciado a esta linha do array.

Repare que das linhas 14 a 19 temos a adição de um elemento link para cada coluna do array para que o título de cada coluna seja transformado em link para que na sequência possamos adicionar um evento ao click destes links. Na linha 29 fechamos a tag <TABLE> e na linha 30 inserimos o conteúdo da variável grid dentro da DIV dvclientes, fazendo com que os contatos retornados do servidor sejam exibidos na nossa pagina HTML.



CUST_NO	CUSTOMER	ADDRESS_LINE1	CITY	COUNTRY	PHONE_NO
1001	Rodrigo Carreiro Mourão	15500 Pacific Heights Blvd.	San Diego	USA	(619) 530-2710
1002	Dallas Technologies	P. O. Box 47000	Dallas	USA	(214) 960-2233
1003	Buttle, Griffith and Co.	2300 Newbury Street	Boston	USA	(617) 488-1864
1004	Central Bank	66 Lloyd Street	Manchester	England	61 211 99 88
1005	DT Systems, LTD.	400 Connaught Road	Central Hong Kong	Hong Kong	(852) 850 43 98
1006	DataServe International	2000 Carling Avenue	Ottawa	Canada	(613) 229 3323
1007	Mrs. Beauvais	P.O. Box 22743	Pebble Beach	USA	
1008	Anini Vacation Rentals	3320 Lawai Road	Lihue	USA	(808) 835-7605
1009	Max	1 Emerald Cove	Turtle Island	Fiji	22 01 23
1010	MPM Corporation	2-64-7 Sasazuka	Tokyo	Japan	3 880 77 19
1011	Dynamic Intelligence Corp	Florhofgasse 10	Zurich	Switzerland	01 221 16 50
1012	3D-Pad Corp.	22 Place de la Concorde	Paris	France	1 43 60 61
1013	Lorenzi Export, Ltd.	Via Eugenia, 15	Milan	Italy	02 404 6284
1014	Dyno Consulting	Rue Royale 350	Brussels	Belgium	02 500 5940
1015	GeoTech Inc.	P.0.Box 702	Den Haag	Netherlands	(070) 44 91 18

Figura 9. jQuery consumindo DataSnap

Nas linhas 31 a 35 temos os códigos necessários para adicionar o evento ao clique dos links contidos nos títulos da tabela. Na linha 31 utilizamos o método find para localizar dentro do conteúdo da DIV dvclientes todas as tags de link (a) do HTML e para cada tag encontrada uma função é adicionada ao evento click. A função está entre as linhas 32 e 35 e quando for executada associará à variável link o elemento que a chamou. Feito isso invocamos novamente o método getclientes (linha 34), porém agora informamos o conteúdo HTML do link clicado, ou seja, o nome da coluna.

Nas linhas 38 a 40 nós utilizamos o método ready para adicionar a chamada ao método getclientes assim que o documento for completamente carregado, simples assim! Para testar a aplicação basta executar o servidor de aplicação e com ele em funcionamento abrir o arquivo clientes.html no browser de sua preferência. Lembre-se que utilizar um servidor Web para abrir o arquivo clientes.html o IIS é uma boa opção. A **Figura 9** exibe a cliente Web em execução.

Conclusão

Neste artigo ainda não tiramos proveito de todo o poder do Delphi 2010 e nem desfrutamos de todas as facilidades que a filosofia REST nos proporciona, mas o que foi visto aqui já é o suficiente para que possamos entender que há um horizonte imenso pela frente. Fica lançado aqui o desafio para que você expanda seus estudos e através de jQuery faça requisições à função GetCustomersJson utilizando os métodos POST, PUT e DELETE criando assim um exemplo de CRUD onde seria possível cadastrar novos clientes, alterar clientes existentes ou mesmo remover um cliente específico.

Você pode ainda mergulhar no mundo Asp.Net e tirar proveitos de todas as novidades do .Net 4.0 na parte de AJAX com o Asp Net Ajax Library. O grande diferencial desta biblioteca é que ela permite a produção de aplicações orientadas a dados (Data-Driven), para serem executadas inteiramente no Web Browser sem a necessidade de nenhum código adicional no servidor, nem a criação de controles de como Textbox, Gridview etc. ●

Links

jQuery
http://jquery.com/

Google - jQuery
http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js

Microsoft - jQuery
http://ajax.microsoft.com/ajax/jquery/jquery-1.4.2.min.js

Blogs do autor
http://rodrigocarreiro.blogspot.com
http://twitter.com/carreiromourao

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Gestures e Multitouch

Crie aplicações em Delphi semelhantes aos aplicativos para iPhone

Do que trata o artigo

Neste artigo iremos explorar Gestures, que permite interpretar e atribuir ações para gestos do usuário na própria tela de maneira incrivelmente simples, criando passo-a-passo uma aplicação com interface similar aos existentes em SmartPhones como o iPhone, que liste e reproduza arquivos mp3 utilizando o componente nativo MediaPlayer. Para a lista de músicas utilizaremos a tecnologia do ClientDataSet para salvar listas em XML e recuperá-las.

Para que serve

A tecnologia de Gestures chega como uma poderosa revolução na interface de entrada de dados, ultrapassa os limites de simples botões a serem clicados, substituindo-os pela liberdade do livre movimento. Com essa tecnologia, conseguimos de maneira eficaz implementar múltiplos comandos em um só componente de forma simples de serem entendidos pelo usuário, tornando os sistemas mais interativos.

Em que situação o tema é útil

A tecnologia de Gestures é hoje utilizada para oferecer ao usuário mais dinamismo, rapidez nos comandos com uma interface amigável, disponibilizada nativamente no Delphi 2010. Com a utilização de Gestures podemos desenvolver telas mais objetivas, que permitam ao usuário uma navegação simples e amigável. Muito vista atualmente em celulares, terminais de aeroportos, totens em shoppings, em telas de pedidos, como restaurantes por exemplo.

Álison Bissoli Dias
bissoli.dias@gmail.com
http://garotodesktop.blogspot.com
Atualmente é analista de sistemas - Mastermaq Softwares. Tem experiência na área de Ciência da Computação, com ênfase em Arquitetura de Sistemas de Computação. Especialista em ERP, Supply Chain, CRM, gestão de requisitos, gestão de processos e implantações de sistemas. Formado em Processamento de Dados em 2004 pela Escola Técnica Juscelino Kubtischeck e cursando ensino Superior Tecnológico em Análise e Desenvolvimento de Sistemas pela Universidade Metodista de Minas.

Resumo do DevMan

Veremos neste artigo como aplicar os recursos disponibilizados no Delphi 2010 para a criação de uma aplicação real que explora uma das sensações do mercado, que é justamente o recurso de Multitouch. Criaremos através de um passo-a-passo um mp3 player, explorando somente componentes nativos, permitindo todas as funcionalidades de um mp3player comum, que somente com a interpretação de gestos, permitirá o controle total do player, juntamente com o ClientDataSet que será utilizado para persistir as listas de músicas em formato XML.

Gestures na informática é a tecnologia de interpretação de gestos, seja movimento das mãos, faciais ou de outras partes do corpo. No Delphi 2010, a tecnologia de Gestures foi implementada pela Embarcadero de modo que o software fosse capaz de reconhecer gestos feitos na tela na área correspondente ao aplicativo. Foram implementadas três tecnologias, sendo elas a Touch, Multitouch e Gestures.

A tecnologia Touch reconhece o toque na área do aplicativo, como se fosse um clique utilizando o mouse na área tocada. MultiTouch reconhece mais de um toque simultaneamente, podendo ser utilizado para mover, rodar e redimensionar componentes utilizando o toque de dois dedos por exemplo.

A tecnologia de Gestures reconhece o gesto, ou seja, o sentido que o gesto foi feito e seu formato. Estas informações capturadas são comparadas com gestos pré-definidos ou gravados nos aplicativos e comparados, respeitando o nível de sensibilidade definido ao gravar o gesto no sistema. Um exemplo seria você fazer o gesto de uma letra "S" na tela e o aplicativo acionar um busca de produtos que iniciem por "S".

Quando falamos de sensibilidade, estamos determinando a tolerância do sistema em aceitar desvios de rota no gesto e ainda o reconhecer. Quanto maior a sensibilidade mais rigoroso o sistema será e mais difícil será para o usuário conseguir reproduzir os gestos para disparar os eventos.

Os gestos funcionam como os eventos conhecidos, ou seja, da mesma forma que implementamos ações para o evento OnClick, podemos implementar códigos para quando um determinado gesto for detectado sobre um objeto. Os gestos deverão ser feitos sobre o componente em que a ação foi implementada para o gesto, a área de reconhecimento, que para um componente será a mesma área que o mesmo ocupa, sendo em primeiro plano ou não, dependendo de configurações que veremos mais a diante.

Nota

Caso seu monitor não suporte o recurso de Gestures, não se preocupe, o movimento dos dedos na tela podem ser substituídos pelo movimento do mouse, de forma que você poderá testar o exemplo deste artigo.

Componentes para suporte a Gestures e Multitouch no Delphi 2010

No Delphi 2010 foi disponibilizada a paleta Gestures, que possui componentes mostrados na **Tabela 1**.

Somente podemos associar gestures a componentes visuais, uma vez que somente componentes visuais possuem área definida em primeiro plano na tela para que sejam interpretados os gestos. Estes componentes possuem a propriedade Touch. Dentro desta propriedade encontramos duas outras propriedades, sendo a primeira GestureManager que trata-se de onde indicamos ao componente qual o seu gestor de movimentos (Gesture Manager). Sem atribuir um Gesture Manager nesta propriedade é impossível implementar o reconhecimento de gestos no componente. Enquanto não houver nada na propriedade Touch.GestureManager, a propriedade Touch.Gestures que é onde associamos os gestos às ações exibe a mensagem Assign GestureManager, indicando que seja inserido um TGestureManager no formulário e informado na propriedade Touch.Gestures.

Após esta atribuição, a propriedade Touch.Gestures de um componente visual poderá ser expandida, abrindo duas ramificações quando possíveis, a Standard (sempre disponível) e a Custom (somente quando o GestureManager possuir algum gesto personalizado). Em Custom serão listados todos os gestos personalizados, para que seja implementada uma ação à mesma. Em Standard será aberta uma série de gestos pré-definidos no próprio Delphi. Para cada propriedade há um CheckBox que indica se o evento está ativo ou não (Enabled), um desenho representativo do gesto e o nome. O nome faz alusão direta ao gesto, por exemplo, a ação da propriedade Left será disparada sempre que for reconhecido um gesto da direita para a esquerda, seguindo uma linha semelhante ao desenho exibido. Na mesma linha de raciocínio a ação associada à propriedade Circle será disparada sempre que um gesto for feito simulando um círculo na área do objeto por exemplo.

As propriedades de Touch.Gestures aceitam implementação direta de código, possibilitando para implementação inserirmos um TActionManager no formulário, criar ações no TActionManager, e as ações criadas no TActionManager serão listadas no ComboBox da propriedade do evento (por exemplo Left) do componente visual.

Outra propriedade de Touch nos componentes visuais é a InterativeGestureOptions, que determina um conjunto de opções usadas para gestos interativos no componente, entre eles a propriedade igoParentPassthrough que possibilita que se um gesto não for reconhecido pelo próprio componente que ocupa a área do gesto em primeiro plano, o gesto será passado para seu parent (container) para ser reconhecido. As propriedades igoPanSingleFingerHorizontal e igoPanSingleFingerVertical tratam da rolagem horizontal e vertical simultaneamente. Ainda em Touch, há a propriedade InterativeGesture, que trata da execução/interpretação dos eventos da **Tabela 2**.

Quando um gesto é interpretado, não é levado em consideração o tamanho ocupado pelo gesto em sua gravação.

TGestureManager	É a classe que gerencia todos os gestos em uma aplicação. Disponibiliza métodos como: gravar gestos em arquivo, carregar gestos, registrar e retirar gestos referenciados a uma coleção dos mesmos.
TGestureListView	Permite que sejam gravados gestos personalizados. É nele que o usuário poderia, por exemplo, criar gestos personalizados para determinadas ações.
TGesturePreview	Utilizado para visualizar um gesto gravado.
TGestureListView	Controle List View para exibir uma série de gestos.

Tabela 1. Componentes da paleta Gestures

igZoom	Gesto de aplicação de zoom, requer o uso de dois dedos.
igPan	Rolagem, somente um dedo.
igRotate	Usado para rotação de componente, requer o uso de dois dedos.
igTwoFingerTap	Toque de dois dedos.
igPressAndTap	Toque de dois dedos, um pressionado e outro para toque.

Tabela 2. Propriedades de Touch.InterativeGesture

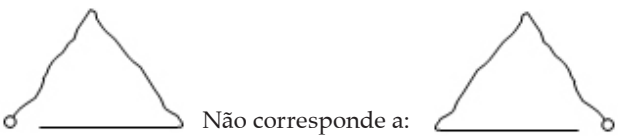
Logo o gesto:

Corresponde ao gesto:

Nota

Considere o ponto ○ como o local inicial do gesto.

Porém, se um gesto é feito em sentido contrário, não é interpretado como sendo o mesmo gesto, mesmo que ele represente a mesma interpretação geométrica.



Não corresponde a:

Criando gestos no Delphi 2010

No Delphi 2010, podemos criar e gravar um gesto novo incluindo o componente TGestureManager. Ao duplo clique sobre o controle, a tela Custom Gestures se abrirá e teremos as opções Create, Delete, Edit, Import e Export. Para criar um novo gesto clique em Create e terá acesso à tela Custom Gesture Designer, conforme mostra a **Figura 1**.

Em Custom Gesture Designer temos todas as ferramentas necessárias para criar, editar, ajustar sensitividade e analisar conflitos. Conflito é uma condição gerada toda vez que o gesto trabalhado em Custom Gesture Designer é similar a um gesto pré-existente, customizado ou standard (padrão). Gestos até mesmo com traços diferentes podem entrar em conflito, bastando que tenham áreas de sensitividade em comum em todo o trajeto. A área de sensitividade é ajustada no controle sensitivity e é representada pelos círculos cinzas que acompanham a linha vetorial do tracejado (azul), tais círculos cinzas aparecem quando ativado o botão "show point sensitivity circles", é interessante essa visualização para saber o grau de dificuldade do usuário para reproduzir o gesto. Para saber onde estão os conflitos acesse o recurso ShowConflictDetails "Detalhar conflitos", e para resolvê-los, ajuste a sensitividade em "sensitivity" ou altere o tracejado.

O quadro pontilhado é a área útil para se desenhar o gesto, note que ao desenhar o traçado são formados pontos vetoriais por todo o caminho, tais pontos podem ser movidos caso necessário. Visualizamos na **Figura 1** cada ponto do vetor do gesto, cada ponto vetorial pode ser movido tanto com o mouse ou caso prefira, alterar a posição do ponto vetorial na tabela de posições. Caso queira complementar o movimento é possível utilizar o botão Insert Point (Ctrl+Ins) e para apagá-los o botão Delete Point (Ctrl + Del). Para alterações minuciosas é possível utilizar o recurso de Zoom dos botões "Zoom in" e "Zoom out".

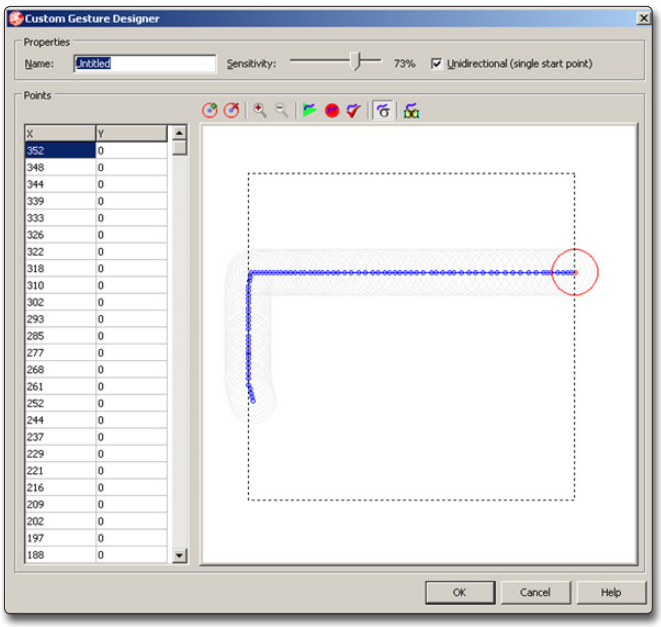


Figura 1. Custom Gesture Designer

Com o gesto criado, simule o gesto com o recurso "Play simulation of gesture", se satisfeito com o resultado tente reproduzi-lo testando em "Test Gesture", cada vez que conseguir, será exibida a frase "Gesture recognized".

Para não gravar e começar a desenhar o gesto novamente utilize "Record Gesture", mas atenção, este botão apaga o estado atual da edição sem solicitar confirmações, portanto, tenha certeza de que quer apagar o desenho atual ao clicar em "Record gesture".

Os gestos pré-existentes (standard) em sua maioria são unidirecionais, somente serão reconhecidos se reproduzidos no mesmo sentido em que foram criados, caso queira que o gesto seja reconhecido independente do sentido desmarque a opção "Unidirectional (single start point)".

Nas **Figuras 2** e **3** respectivamente, o círculo é o ponto inicial do gesto, a linha em si representa o gesto que deve ser feito e a área ao redor do gesto ilustra o que seria considerado válido caso fosse desviado. Nas imagens podemos observar que há áreas em comum nos gestos A e B, o que possibilita que um gesto seja interpretado como outro, neste caso aumentar a sensitividade é uma boa solução para resolver este problema, outra solução é não criar gestos parecidos para o mesmo componente.

Quando houver este tipo de problema, o botão Show Conflict Details do Custom Gesture Manager ficará ativo e você poderá conferir com quais gestos o seu gesto que está sendo criado está conflitante para que o mesmo seja alterado.

Caso queira gravar o gesto criado, dê um nome em Name e clique em Ok, caso contrário, clique em Cancel. Ao tentar gravar um gesto conflitante uma tela de confirmação será exibida.

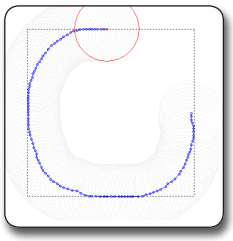


Figura 2. Reprodução do gesto A

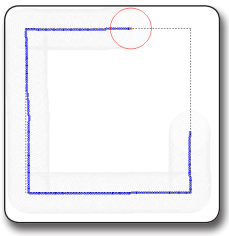


Figura 3. Reprodução do gesto B

O componente TMediaPlayer

O componente TMediaPlayer é encontrado na paleta System e proporciona uma interface de controle de mídia (MCI). Com ele pode-se controlar dispositivos multimídia como um CD-ROM ou VCR. Embora tenha vários botões, estes não se separam do próprio componente. Na **Figura 4** você observa um exemplo dos controles disponíveis no componente. A **Tabela 3** descreve suas principais propriedades e a **Tabela 4** seus principais métodos.

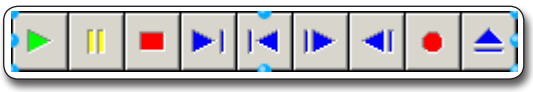


Figura 4. Componente TMediaPlayer

Iniciando o projeto

Vamos colocar as mãos na massa a partir de agora para criarmos nosso Player. Abra o Delphi 2010 e crie um novo projeto, acessando File > New > VCL Forms Application. Ao Form1, modifique as propriedades Name para frmMp3Player, Caption para "MP3 Player ClubeDelphi", Position para poDesktopCenter e altere as dimensões, informando 680 para Width e 300 em Height.

Vamos salvar nosso projeto, acesse File > Save e informe o nome ufrmMp3Player.pas para a unit do projeto e o nome Mp3Player.dproj para o projeto. Selecione a paleta Data Access, adicione um componente ClientDataSet e renomeie-o para cdsPlaylist. Acesse o Fields Editor com um duplo clique sobre o componente. Nesta janela do Fields Editor criaremos a estrutura da tabela que guardará as informações da nossa PlayList. Vamos armazenar no componente as seguintes informações: título da faixa, autor, álbum e endereço.

Preparando o ClientDataSet para listar as músicas

Muito utilizado como provedor de dados, nos habituamos a encontrar este componente ligado à DataSetProviders, agora trabalharemos com ele em modo "off-line", armazenando nossas listagens de músicas em arquivos XML. Criaremos vários campos no ClientDataSet para armazenar todas as informações dos arquivos selecionados pelo usuário, e usaremos o comando CreateDataSet para que ele funcione como uma tabela de memória, desligado de qualquer fonte de dados. A partir daí poderemos alimentar o ClientDataSet com os dados das músicas.

Para criarmos os campos, clique com o botão direito em uma área livre do Fields Editor e clique em New Field. Devemos colocar aqui as mesmas informações que colocaríamos se estivéssemos criando uma tabela em um BD, sendo a propriedade Name o nome do respectivo campo, Type refere-se ao tipo do dado e em Size é definido o tamanho. Como trabalharemos com a gravação de dados em XML para recuperarmos novamente, deixaremos a propriedade FieldType de todos os campos como Data, pois todos serão armazenados fisicamente. Crie os campos conforme a **Tabela 5.**

Após ter criado os campos insira um DataSource, que servirá para realizar a comunicação dos dados com os demais componentes visuais, no nosso caso, um DBGrid. Renomeie-o para dtsMusicas, e informe à propriedade DataSet o cdsPlaylist.

TrackLenght	Informa a duração da faixa.
TrackPosition	Informa a posição inicial da faixa.
Error	Informa o código de erro.
ErrorMessage	Informa o texto de erro.
Start	Posição inicial ao iniciar a mídia atual.
Lenght	Informa a duração da mídia do dispositivo.
Tracks	Informa a quantidade de mídias tocáveis do dispositivo.
Frames	Informa a quantidade de frames avançados ou retornados.
Mode	Indica o estado do dispositivo acessado.
Position	Informa a posição atual do progresso da faixa carregada.
Wait	Trava a aplicação enquanto a mídia não for concluída.
StartPos	Especifica a partir de qual posição será tocada a mídia.
EndPos	Especifica até qual posição será tocada a mídia.
TimeFormat	Descreve o formato usado para especificar a posição.
ColoredButtons	Determina quais botões possuem cor.
EnabledButtons	Determina quais botões são utilizáveis.
VisibleButtons	Determina quais botões são visíveis.
AutoEnable	Permite que o próprio componente ative/desative os botões que o compõe automaticamente.
DeviceType	Especifica um tipo de dispositivo para abrir o Media Player.
FileName	Especifica o nome do arquivo aberto ou salvo.
Shareable	Determina se mais de um aplicativo pode compartilhar um dispositivo multimídia.

Tabela 3. Propriedades do componente TMediaPlayer

Open	Abre um dispositivo/acessa o arquivo informado no FileName.
Close	Fecha um dispositivo.
Play	Reproduz a mídia do dispositivo.
Stop	Cancela a reprodução ou gravação.
Pause	Alterna o dispositivo entre aberto/desligado.
Step	Avança a quantidade de frames determinada na propriedade Frames.
Back	Retorna a quantidade de frames determinada na propriedade Frames.
Previous	Retorna a posição para o início da faixa atual / retorna para a faixa anterior.
Next	Avança para a próxima faixa da mídia.
StartRecording	Inicia gravação a partir da posição atual ou da posição especificada em StartPos.
Eject	Retirada da mídia física carregada do dispositivo multimídia aberto.
Save	Salva a mídia carregada em caminho especificado na propriedade FileName.
PauseOnly	Pausa a mídia que se está sendo reproduzida.
Resume	Volta a reproduzir a partir da posição pausada.

Tabela 4. Principais métodos do componente TMediaPlayer

Name	Type	Size	FieldType	DisplayLabel
strEndereco	String	500	Data	Endereço
strTitulo	String	60	Data	Título
strArtista	String	60	Data	Artista
strAlbum	String	60	Data	Álbum

Tabela 5. Definição dos campos a serem criados no ClientDataSet

Criando a estrutura da tela

Para iniciar a criação da nossa interface, vamos inserir dois Panels, e renomeá-los respectivamente para pnlControles e pnlLista, em seguida altere suas propriedades Align para alTop e alClient. Insira um componente Image dentro do pnlLista, renomeie-o para imgBibliotecaAdicionarExcluir, modifique seu Align para alRight, no espaço ainda vago do pnlLista insira um DBGrid e renomeie-o para dbgPlayList, ligue a propriedade DataSource ao dtsMusicas e altere seu Align para alClient. Para inserir as colunas referentes aos campos do ClientDataSet dê duplo clique no dbgPlayList, onde teremos uma janela semelhante ao Fields Editor, porém para a edição de colunas, clique sobre o botão Add All Fields.

Insira em pnlControles um Image e altere suas propriedades Name para imgControle e Align para alClient. Na propriedade Picture do componente imgControle selecione a imagem topo_Menu de controle.jpg, e na mesma propriedade do componente imgBibliotecaAdicionarExcluir selecione a imagem esquerda_biblioteca_adicionar_excluir.jpg.

Nota

As imagens utilizadas no exemplo estão disponíveis para download juntamente com o projeto do artigo.

Insira três Labels no pnlControles nomeie-os sequencialmente para lbltitulo, lblArtista e lblAlbum, em seguida, posicione-os em frente de título, artista e álbum. Todos estes Labels devem possuir o Caption "...". Abaixo de álbum, insira um componente ProgressBar e altere suas propriedades Name para pBar e Smooth para True. Neste momento sua tela deve estar como ilustrado na **Figura 5**.

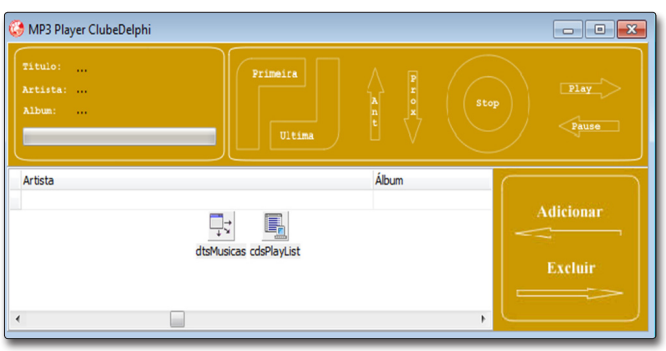


Figura 5. Estrutura da tela em tempo de design

Insira um componente MediaPlayer e modifique suas propriedades Name para mplMediaPlayer e Visible para False, este será o responsável pelas execuções das mídias. Note que a propriedade Visible foi setada como False porque o usuário não precisará clicar diretamente no componente, atribuiremos eventos às imagens da tela para que façam o trabalho dos botões.

Insira um componente GestureManager, da guia Gestures, e nomeie-o para gmPrincipal, em seguida, selecione os dois Images e expanda a propriedade Touch (+) atribuindo o gmPrincipal à propriedade GestureManager.

Procedures de apoio

Seguindo as boas normas de refatoração, os blocos de código que seriam utilizados em várias partes do programa se tornaram funções ou procedures separadas. Sendo assim teremos quatro métodos que serão apresentados nas listagens a seguir. Faça suas implementações e pressione simultaneamente as teclas Ctrl + Shift + C para que sejam declarados suas assinaturas na sessão private.

A procedure prepararClientDataSet exibida na **Listagem 1** verifica se o ClientDataSet não está ativo, sendo assim, limpamos sua propriedade FileName para termos a certeza de que não existe nenhuma lista salva. Em seguida é executado o método CreateDataSet, que automaticamente prepara o ClientDataSet para armazenamento de dados em memória.

Declararemos a procedure alimentaCaptions (**Listagem 2**) que conforme o próprio nome sugere, atribuirá o conteúdo dos campos aos respectivos Labels.

Listagem 1. Preparando o ClientDataSet

```
procedure TfrmMp3Player.prepararClientDataSet;
begin
 if not cdsPlaylist.Active then
 begin
  cdsPlaylist.FileName := '';
  cdsPlaylist.CreateDataSet;
 end;
end;
```

Listagem 2. Configurando os Labels

```
procedure TfrmMp3Player.alimentaCaptions;
begin
  lblTitulo.Caption  := cdsPlaylist.FieldValues['strTitulo'];
  lblArtista.Caption := cdsPlaylist.FieldValues['strArtista'];
  lblAlbum.Caption   := cdsPlaylist.FieldValues['strAlbum'];
end;
```

Também criamos a função MediaPlayerPreparado (**Listagem 3**). Na Linha 04 é verificado o estado do componente MediaPlayer, checando se o mesmo está aberto e fechando-o em seguida. A seguir na Linha 06 é validado o conteúdo do campo strEndereco, caso este seja nulo, a execução será desviada para a Linha 16, onde teremos a limpeza do arquivo dentro do componente, caso contrário, é checada existência do arquivo no disco (Linha 08) para posteriormente ser carregado dentro do componente (Linha 10). Por último o status do componente MediaPlayer é novamente verificado, e não estando pausado o mesmo será aberto (Linhas 11 e 12). Note que caso o arquivo seja encontrado a nossa função retornará Verdadeiro (Linha 13), não encontrando o retorno será o definido logo no início, na Linha 03.

O último de nossos quatro métodos será a procedure executarMusicaSelecionada (**Listagem 4**). Perceba que efetuamos uma chamada para nossa função MediaPlayerPreparado, pois como visto na **Listagem 3,** esta será responsável pelo carregamento do arquivo dentro do componente MediaPlayer. Aqui o nosso ProgressBar tem atribuição de seu valor inicial como 0 e valor final conforme o tamanho do arquivo carregado. Veja que temos no mesmo código a execução da música no método Play e posteriormente a chamada da procedure alimentaCaptions que exibe as informações dos campos nos Labels.

Finalmente, insira no projeto um Timer e renomeie-o para tmrProgresso. No evento OnTimer faça o código a seguir, responsável por incrementar a posição do ProgressBar de acordo com o progresso da música em execução:

```
procedure TfrmMp3Player.tmrProgressoTimer(Sender: TObject);
begin
 if (pBar.Max <> 0) and (mplMediaPlayer.Mode = mpPlaying) then
   pBar.Position := mplMediaPlayer.Position;
end;
```

Salvando em XML e abrindo a listagem de músicas

Para que seja possível utilizar as janelas padrões do Windows para abrir e salvar arquivos, insira um OpenDialog, altere sua propriedade Name para dlgAbrir e em Options marque ofAllowMultiSelect para que possamos selecionar mais de um arquivo por vez. Este componente será utilizado tanto para selecionar músicas quanto para selecionar XMLs de listagens de músicas. Insira um SaveDialog que usaremos para salvar a lista em XML, renomeie-o para dlgSalvar e configure sua propriedade Filter atribuindo o texto "Arquivos XML" para Filter Name e *.xml para Filter, deste modo manipularemos apenas arquivos XML.

Insira a seguir ActionList para guardar todas as ações em grupos, organizando assim nosso projeto. Atribua o nome de actBiblioteca, e com um duplo clique sobre o componente crie as ações actAbrirLista de Caption "Abrir lista" e actSalvarLista de Caption "Salvar lista". Para melhorar a organização das ações digite na propriedade Category o texto "Biblioteca", assim as ações serão automaticamente classificadas nesta categoria. Feito isso implemente os códigos nas respectivas ações criadas, para a execução dos componentes SaveDialog e OpenDialog, conforme ilustra a **Listagem 5.**

Criada as ações, insira um PopupMenu nomeando-o para popLista. Dê um duplo clique sobre ele e crie dois novos itens, apontando na propriedade Actions cada uma das ações criadas. Para que seja chamado como menu de contexto do DBGrid, simplesmente selecione-o na propriedade PopupMenu.

Adicionando e excluindo músicas a sua PlayList

No ActionList, crie as ações actAdicionar de Caption Adicionar e actExcluir de Caption Excluir. Selecione a Category Biblioteca para ambas. Antes de prosseguirmos com a codificação, iremos declarar uma variável de tipo packed Record, que será utilizada para armazenamento temporário das informações obtidas em cada um dos arquivos de mídia. Declare-o no começo do formulário como a **Listagem 6**.

Listagem 3. Declaração da função MediaPlayerPreparado

```
01 function TfrmMp3Player.MediaPlayerPreparado: boolean;
02 begin
03   Result := False;
04   if mplMediaPlayer.Mode = mpOpen then
05     mplMediaPlayer.Close;
06   if not cdsPlaylist.FieldByName('strEndereco').IsNull then
07   begin
08     if FileExists(cdsPlaylist.FieldByName('strEndereco').AsString) then
09     begin
10       mplMediaPlayer.FileName :=
         cdsPlaylist.FieldByName('strEndereco').AsString;
11       if mplMediaPlayer.Mode <> mpPaused then
12         mplMediaPlayer.Open;
13       Result := True;
14     end
15     else
16       mplMediaPlayer.FileName := '';
17   end;
18 end;
```

Listagem 4. Executando a música

```
procedure TfrmMp3Player.executarMusicaSelecionada;
begin
  if not MediaPlayerPreparado then
    exit;
  pBar.Position := 0;
  mplMediaPlayerPplay;
  pbar.max := mplMediaPlayer.Length;
  alimentaCaptions;
end;
```

Listagem 5. OnExecute de actAbrirLista e actSalvarLista

```
procedure TfrmMp3Player.actAbrirListaExecute(Sender: TObject);
begin
  dlgabrir.Filter := 'Arquivos XML|*.xml';
  if dlgabrir.Execute() then
    cdsPlaylist.LoadFromFile(dlgabrir.FileName);
end;

procedure TfrmMp3Player.actSalvarListaExecute(Sender: TObject);
begin
  if dlgSalvar.Execute() then
  begin
    if UpperCase(copy(dlgsalvar.FileName,length(dlgsalvar.FileName)-3,4))
      <> '.XML' then
      dlgsalvar.FileName := dlgsalvar.FileName + '.xml';
    cdsPlaylist.SaveToFile(dlgsalvar.FileName);
  end;
end;
```

Listagem 6. Declaração do tipo packed record

```
 type
 TID3 = packed record
   TAG: array [0..2] of AnsiChar;
   Titulo: array [0..29] of AnsiChar;
   Artista: array [0..29] of AnsiChar;
   Album: array [0..29] of AnsiChar;
   Ano: array [0..3] of AnsiChar;
   Commentarios: array [0..29] of AnsiChar;
   Genero: byte;
 end;

 var
  frmMp3Player: TfrmMp3Player;
```

A seguir implemente o evento OnExecute do actAdicionar conforme a **Listagem 7**. Primeiramente executamos nosso método de preparação do ClientDataSet para a atribuição da PlayList (Linha 07), perceba que em seguida é solicitada a escolha de arquivos mp3 na Linha 09. Na Linha 11 temos a inicialização de um contador, responsável por percorrer todos os itens de mp3 selecionados.

Note que entre as Linhas 13 à 19 é declarada uma variável do tipo FileStream, que carrega as informações do arquivo encontrado para dentro do packed Record ID3TAG através do método Read (Linha 16), posteriormente liberando o FileStream da memória através do método Free (Linha 18). A seguir entre as Linhas 20 à 34 é checado o atributo Tag, para só então atribuir as informações encontradas e armazenadas no packed Record para dentro de nosso ClientDataSet.

Listagem 7. Evento OnExecute da ação actAdicionar

```
01 procedure TfrmMp3Player.actAdicionarExecute(Sender: TObject);
02 var
03   I: integer;
04   id3tag: TID3;
05   mp3file: Tfilestream;
06 begin
07   prepararClientDataSet;
08   dlgabrir.Filter := 'Arquivos MP3|*.mp3';
09   if dlgAbrir.Execute then
10   begin
11     for I := 0 to dlgAbrir.Files.count - 1 do
12     begin
13       mp3file := TFileStream.Create(dlgAbrir.Files.Strings[I],
       fmOpenRead);
14       try
15         mp3file.position := mp3file.size - 128;
16         mp3file.Read(id3tag, SizeOf(id3tag));
17       finally
18         mp3file.free;
19       end;

20       cdsPlaylist.Append;
21       cdsPlaylist.FieldValues['strEndereco'] :=
         dlgAbrir.Files.Strings[I];
22       if id3tag.TAG <> 'TAG' then
23       begin
24         cdsPlaylist.FieldByName('strTitulo').AsString := 'N/A';
25         cdsPlaylist.FieldByName('strArtista').AsString := 'N/A';
26         cdsPlaylist.FieldByName('strAlbum').AsString := 'N/A';
27       end
28       else
29       begin
30         cdsPlaylist.FieldByName('strTitulo').AsString := id3tag.Titulo;
31         cdsPlaylist.FieldByName('strArtista').AsString :=
           id3tag.Artista;
32         cdsPlaylist.FieldByName('strAlbum').AsString := id3tag.Album;
33       end;
34       cdsPlaylist.post;
35     end;
36   end;
37 end;
```

Agora a faça a implementação do evento responsável por excluir a PlayList através do evento OnExecute no actExcluir, conforme o código que se segue:

```
procedure TfrmMp3Player.actExcluirExecute(Sender: TObject);
begin
  prepararClientDataSet;
  if cdsPlaylist.RecordCount <> 0 then
    cdsPlaylist.Delete;
end;
```

Eventos para controle das músicas

Criaremos agora os eventos de controle das músicas, para isso definiremos no actBiblioteca novas ações. Adicione as ações actPlayerExecute, actPlayerPause, actPlayerStop, actPlayerProximo, actPlayerAnterior, actPlayerPrimeira e actPlayerUltima e defina respectivamente os Captions para Play, Pause, Stop, Próximo, Anterior, Primeira e Última. Por fim, crie como anteriormente uma nova categoria para estas ações classificando-as como "Player". Suas codificações estão exibidas na **Listagem 8**.

Atribuindo os eventos aos gestures dos componentes

Todos os nossos eventos da Category Player serão chamados por Gestures, para tanto, atribua as ações de imgBibliotecaAdicionarExcluir conforme mostra a **Figura 6**. Finalmente, para imgControle, atribua as ações conforme a **Figura 7**.

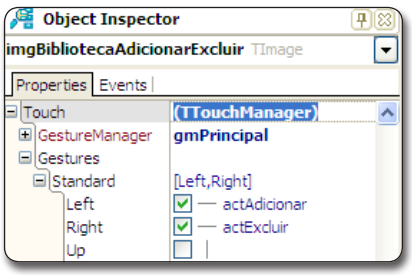


Figura 6. Ações do componente imgBibliotecaAdicionarExcluir

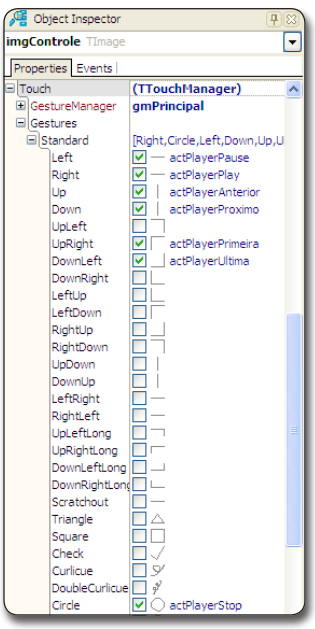


Figura 7. Ações do componente imgControle

Interface mais agradável

Para deixar a tela mais funcional para o usuário, faremos com que o usuário execute a música ao realizar um duplo clique sobre o Grid. Para isto, implemente o código a seguir no evento OnDblClick do DBGrid:

```
procedure TfrmMp3Player.dbgPlayListDblClick(Sender: TObject);
begin
  executarMusicaSelecionada;
end;
```

Listagem 8. Codificações das ações

```
[actPlayerExecute]
procedure TfrmMp3Player.actPlayerPlayExecute(Sender: TObject);
begin
  executarMusicaSelecionada;
end;

[actPlayerPause]
procedure TfrmMp3Player.actPlayerPauseExecute(Sender: TObject);
begin
  mplMediaPlayer.StartPos := mplMediaPlayer.Position;
  if mplMediaPlayer.Mode = mpPlaying then
  mplMediaPlayer.PauseOnly;
end;

[actPlayerStop]
procedure TfrmMp3Player.actPlayerStopExecute(Sender: TObject);
begin
  if not MediaPlayerPreparado then
  exit;
  mplMediaPlayer.Stop;
  mplMediaPlayer.FileName := '';
  pbar.Position := 0;
  pbar.max := mplMediaPlayer.Length;
end;

[actPlayerProximo]
procedure TfrmMp3Player.actPlayerProximoExecute(Sender: TObject);
begin
  if cdsPlaylist.Active then
    cdsPlaylist.Next;
  executarMusicaSelecionada;
end;

[actPlayerAnterior]
procedure TfrmMp3Player.actPlayerAnteriorExecute(Sender: TObject);
begin
  if cdsPlaylist.Active then
    cdsPlaylist.Prior;
  executarMusicaSelecionada;
end;

[actPlayerPrimeira]
procedure TfrmMp3Player.actPlayerPrimeiraExecute(Sender: TObject);
begin
  if cdsPlaylist.Active then
    cdsPlaylist.First;
  executarMusicaSelecionada;
end;

[actPlayerUltima]
procedure TfrmMp3Player.actPlayerUltimaExecute(Sender: TObject);
begin
  if cdsPlaylist.Active then
    cdsPlaylist.last;
  executarMusicaSelecionada;
end;
```

E por último, para que o Grid possa organizar as informações com um clique no título da coluna, faça a seguinte codificação em seu evento OnTitleClick:

```
procedure TfrmMp3Player.dbgPlayListTitleClick(Column: TColumn);
begin
  prepararClientDataSet;
  cdsPlaylist.IndexFieldNames := column.FieldName;
end;
```

Tendo realizado toda a codificação, e associado os Actions aos eventos Gestures, basta repetir o gesto indicado pelas imagens para executar as ações, sem utilização de botões e proporcionando que o usuário tenha uma tela completamente limpa e dinâmica. A **Figura 8** mostra a aplicação final em execução.

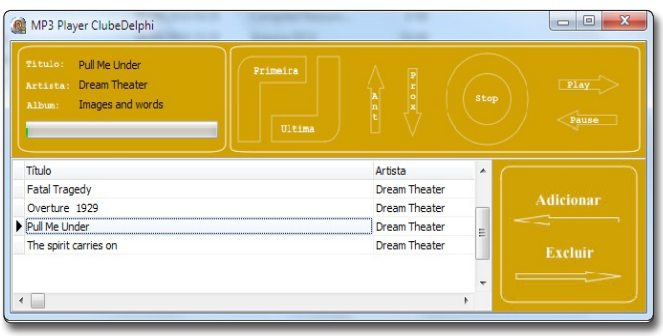


Figura 8. Aplicação em execução

Conclusão

A tendência é que as aplicações Desktop sejam tão intuitivas quanto às aplicações Mobile. Experimente incluir as facilidades dos Gestures em suas aplicações, não tema em inovar, não se justifica mais colocar botões e mais botões nas telas a ponto de deixá-las confusas e carregadas. Experimente criar Gestures personalizados no contexto de seus sistemas, utilizá-lo como um tipo de chave de acesso, realizando gestos determinados pelo usuário, no lugar de uma string de senha para seu sistema. Use e abuse ●

Links

Blog do autor
http://garotodesktop.blogspot.com

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre técnicas que poderão aumentar a qualidade do desenvolvimento de software

Padrão Observer

Não ligue para mim. Eu ligo para você!

Resumo do DevMan

Sistemas modernos precisam deixar o usuário informado sobre as mudanças que ocorrem em seu ambiente em tempo real. É isto que se espera de um sistema que se diga integrado. Um bom exemplo para isto é um sistema que possua controle de estoque. Normalmente estes sistemas possuem um algoritmo para fazer o balanço do estoque da empresa. Quando o balanço é iniciado muitas funções do sistema ligadas ao movimento de estoque devem ficar indisponíveis até que o balanço termine. Muitas vezes este controle não é feito e fica sendo responsabilidade do usuário cuidar para que nenhuma venda ocorra durante o processo de balanço. Neste artigo você vai aprender o que fazer para que as partes do seu sistema interajam entre si de maneira elegante, através do padrão Observer.

Do que trata o artigo

Aprenda como integrar as partes do seu sistema de maneira sofisticada. Através do uso das interfaces o Padrão Observer permite que partes distintas se comuniquem sem que uma parte precise conhecer a outra.

Para que serve

Com o Padrão Observer você consegue criar um sistema realmente integrado, pois cada parte do seu sistema será notificado sempre e somente se necessário sobre as mudanças de outras partes. Integração quer dizer que a aplicação funciona de maneira diferente de acordo com o que está acontecendo em cada pedaço do sistema.

Em que situação o tema é útil

A palavra chave é integração. Quando se deseja integrar partes do seu sistema sem que seja criado forte acoplamento entre elas, de tal forma que as partes envolvidas se interajam apenas quando for necessário e não em determinados espaços de tempo.

Rafael Stavarengo
faelsta@gmail.com
http://www.devmedia.com.br/stavarengo
Programador-analista da Cheina Informática. Trabalha com Delphi, JME e PHP + ZendFramework. Formando em Técnologia de Análise e Desenvolvimento de Sistemas pela UNIPAR.

O padrão Observer garante que as partes específicas de um sistema sejam notificadas de mudanças que acontecem em alguma área do sistema, porém, sem que estas duas partes saibam muito uma da outra, ou seja, a tela de vendas pouco ou nada saberia a respeito do objeto Produtos, mas de alguma maneira seria informada quando o objeto Produtos iniciar a rotina de balanço de estoque.

Entenda o que é um sistema integrado

Imaginando o cenário proposto no quadro Resumo do Dev Man, uma das soluções para que o módulo de vendas fique ciente de que o processo de balanço está ativo, é implementar uma verificação pelo uso do TTimer, onde em um período estabelecido, o sistema verificaria se o balanço está sendo feito. Apesar desta ser uma solução comum, não é a mais indicada visto que um número excessivo de solicitações estarão acontecendo desnecessariamente, além do fato de que existe um intervalo de tempo entre uma solicitação e outra. Imagine a seguinte situação:

- O usuário Marcos abre o formulário de venda;
- A usuária Jaqueline, abre a janela de balanço, mas ainda não o inicia;
- O TTimer, no computador do Marcos, faz uma verificação para garantir que nenhum balanço foi iniciado;
- Jaqueline inicia o balanço;
- Marcos, salva a venda, o que não devia ser permitido porque a Jaqueline iniciou o balanço;
- O TTimer, no computador do Marcos, faz a verificação novamente e agora como o balanço foi iniciado, desabilita o uso da venda para o Marcos, porém, faz isto tarde demais, pois Marcos já fez uma venda quando existia um balanço em andamento.

Neste simples exemplo você pode ver como é frágil um sistema cujas partes são integradas usando um TTimer. Você pode diminuir a possibilidade disso acontecer diminuindo o intervalo de tempo, mas isto traz outra consequência: com o uso excessivo de TTimers você faz seu sistema parecer mais lento, já que um processamento extra é executado a cada X milissegundos. Pense no TTimer como um objeto "burro", pois ele não sabe quando a informação mudou e faz verificações repetidas vezes mesmo que nada tenha mudado. Pior do que usar TTimer para integrar as partes do sistema é não integrar o sistema.

O Padrão Observer

Princípio de orientação a objetos

Busque projetos levemente ligados entre objetos que interagem.

Integrar as partes do sistema não é uma tarefa simples, principalmente quando você tem um sistema que funciona em rede com múltiplos usuários logados ao mesmo tempo. O problema do processo de integração é fazer com que as diversas partes do sistema conversem entre si. Isto é realmente um problema, pois em um mundo orientado a objetos os sistemas devem ser construídos de maneira que uma parte conheça o mínimo ou até mesmo não conheça a outra parte. Então fica a dúvida: como a parte X pode saber se algo mudou na parte Y se o X não pode conhecer o Y? Colocar um TTimer seria declarar descaradamente que as duas partes se conhecem, pois a parte com o TTimer terá que conhecer os detalhes da parte que será verificada. Felizmente outros programadores já passaram por este problema e documentaram a solução que encontraram. Para esta solução eles deram o nome de Padrão Observer.

O exemplo

Para explicar o Padrão Observer, vou criar durante este artigo uma pequena aplicação de exemplo. Esta aplicação terá duas funções: vender e fazer balanço do estoque. Não será muito complexo, pois o objetivo do artigo não é ensinar a criar uma rotina de venda ou de balanço, mas sim de ensinar como fazer para que estas e outras partes do sistema possam se comunicar para se tornar uma integração de verdade.

Obviamente esta será uma aplicação orientada a objetos, então os objetos envolvidos no problema devem ser encontrados antes de começar a aplicação. O problema que precisa ser solucionado é: fazer vendas e balanço de estoque. Eu encontrei dois objetos envolvidos neste problema, são eles: TVenda e TBalanco. Eu pensei na classe TVenda para solucionar o problema de se fazer vendas e na classe TBalanco para fazer o balanço de estoque da empresa. Não posso me esquecer também do formulário principal da aplicação, pois sem ele o usuário não consegue usar o sistema. Então são três classes, contando com o formulário principal, para resolver o problema de venda e balanço de estoque: TVenda, TBalanco e TFormPrincipal. Veja na **Figura 1** o diagrama de classes da aplicação.

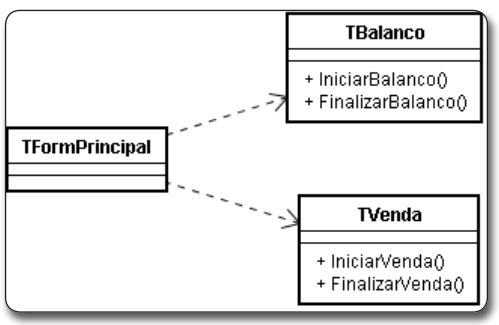


Figura 1. As classes do sistema de exemplo

Você pode notar duas informações importantes no diagrama de classes: o formulário principal depende das classes TBalanco e TVenda para existir e as classes TBalanco e TVenda não se conhecem. O formulário principal será tão simples quanto o exemplo que está sendo criado. Nele existirão quatro botões: um para iniciar a venda, um para finalizá-la, um para iniciar o balanço e um último para finalizá-lo, além de dois componentes TLabel para indicar se existe alguma venda ou balanço iniciado. Veja na **Figura 2** como é o layout do formulário principal da aplicação.

Codificando

Inicie um novo projeto no Delphi 2010 através do menu File > New > VCL Forms Application. Isto fará com que o Delphi crie uma nova unit já com um formulário criado. Salve o projeto com o nome Exemplo.dpr no local que achar melhor. Em seguida salve a unit do formulário com o nome Principal.pas. Depois de salvar os arquivos corretamente, adicione e configure os componentes conforme aparece na **Tabela 1**. Em seguida tente deixar os componentes como apresentado na **Figura 2**.

Antes de começar a escrever as primeiras linhas de código na classe TFormPrincipal é necessário criar as classes TVenda e

TBalanco, pois a classe TFormPrincipal depende diretamente destas classes para funcionar e por este motivo não é possível implementar a classe TFormPrincipal antes destas duas classes. Veja na **Listagem 1** o código da classe TVenda e na **Listagem 2** a classe TBalanco.

Figura 2. O formulário principal

Componente	Propriedade	Valor
TForm	Name	FormPrincipal
TButton	Name	BotaoIniciarVenda
	Caption	Iniciar venda
TButton	Name	BotaoFinalizarVenda
	Caption	Finalizar venda
TButton	Name	BotaoIniciarBalanco
	Caption	Iniciar balanço
TButton	Name	BotaoFinalizarBalanco
	Caption	Finalizar balanço
TLabel	Name	LabelStatusVenda
	Caption	Nenhuma venda iniciada
	Font.Style	[fsBold]
TLabel	Name	LabelStatusBalanco
	Caption	Nenhum balanço iniciado
	Font.Style	[fsBold]

Tabela 1. Componentes do formulário principal

Entenda como a classe TVenda funciona. Esta classe possui apenas dois métodos: IniciarVenda e FinalizarVenda. Eles estão declarados respectivamente nas linhas 8 e 9 da **Listagem 1**. Você precisa ter em mente a simplicidade desta aplicação. O método IniciarVenda é apenas uma simulação, ou seja, neste exemplo ele representa um método mais complexo que existe em uma aplicação do mundo real para que um usuário possa vender algo. Essa regra de simplicidade vale para todas as classes do exemplo criado neste artigo.

Com isto explicado, veja o que acontece no método IniciarVenda. A implementação dele está entre as linhas 13 e 20 da **Listagem 1**. Antes de tudo, na linha 15 é feito um teste para garantir que não existe nenhuma venda iniciada, pois não faz sentido iniciar duas vendas ao mesmo tempo. Se já existir uma venda iniciada, então nada é feito. Por outro lado, se não existe nenhuma venda então o valor da variável VendaIniciada é alterado para True, indicando de que uma venda já existe – a variável VendaIniciada está declarada na linha 6. Em seguida na linha 18, uma mensagem é apresentada para o usuário informando que a venda foi iniciada. Repare que esta mensagem não aparecerá se a venda já estava iniciada.

Listagem 1. Classe TVenda

```
1  unit Venda;

2  interface

3  type
4   TVenda = class
5   private
6    VendaIniciada: Boolean;
7   public
8    procedure IniciarVenda;
9    procedure FinalizarVenda;
10 end;

11  implementation

12  uses Dialogs;

13  procedure TVenda.IniciarVenda;
14  begin
15   if not VendaIniciada then
16   begin
17    VendaIniciada := True;
18    ShowMessage('Venda iniciada!');
19   end;
20  end;

21  procedure TVenda.FinalizarVenda;
22  begin
23   if VendaIniciada then
24   begin
25    VendaIniciada := False;
26    ShowMessage('Venda finalizada!');
27   end;
28  end;

29  end.
```

O método FinalizarVenda é muito parecido, porém faz o contrário do método IniciarVenda. Veja a implementação dele entre as linhas 21 e 28. Primeiro, na linha 23 é feito um teste para garantir que existe uma venda para finalizar. Se nenhuma venda existir nada é feito, pois não há vendas para finalizar. Mas, se alguma venda existir então o valor da variável VendaIniciada é alterado novamente, só que desta vez para False, indicando que não existe uma venda em andamento. Em seguida na linha 26 uma mensagem é mostrada para o usuário confirmando que a venda foi finalizada. Repare que assim como a mensagem do método IniciarVenda, esta também só aparecerá se existir alguma venda para ser finalizada. Agora veja a **Listagem 2**, nela está a classe TBalanco.

Assim como a classe TVenda, a classe TBalanco é bem simples. Na verdade elas são quase idênticas. A classe TBalanco também possui dois métodos, um para iniciar e outro para finalizar o balanço, assim como a classe TVenda possui um para iniciar e finalizar. Veja nas linhas 8 e 9 da **Listagem 2**. Estes dois métodos são similares aos da classe TVenda. Se você entendeu como funciona a classe TVenda você também já entendeu como funciona a classe TBalanco.

O formulário principal

Com as classes TVenda e TBalanco funcionando, já é possível codificar a classe TFormPrincipal. Este formulário terá quatro botões, uma para cada um dos métodos existentes nas duas classes que criei. Veja na **Listagem 3** a implementação da classe TFormPrincipal.

Listagem 2. Classe TBalanco

```
1  unit Balanco;

2  interface

3  type
4   TBalanco = class
5   private
6    BalancoIniciado: Boolean;
7   public
8    procedure IniciarBalanco;
9    procedure FinalizarBalanco;
10  end;

11  implementation

12  uses Dialogs;

13  procedure TBalanco.IniciarBalanco;
14  begin
15   if not BalancoIniciado then
16   begin
17    BalancoIniciado := True;
18    ShowMessage('Balanco Iniciado!');
19   end;
20  end;

21  procedure TBalanco.FinalizarBalanco;
22  begin
23   if BalancoIniciado then
24   begin
25    BalancoIniciado := False;
26    ShowMessage('Balanco finalizado!');
27   end;
28  end;

29  end.
```

Você viu no diagrama de classes da **Figura 1** que a classe TFormPrincipal depende das classes TVenda e TBalanco. Esta informação é comprovada agora, pois na **Listagem 3** você observa nas linhas 2 e 3 que TFormPrincipal tem duas variáveis, uma de cada classe que ela depende. É por isto que posso dizer que TFormPrincipal é dependente destas duas classes, pois se uma destas for removida, TFormPrincipal para de funcionar – ou seja, ela é altamente dependente de TVenda e TBalanco. Da maneira como o código está escrito é possível dizer que o relacionamento destas duas classes com TFormPrincipal é um relacionamento do tipo Composição, mas isto já é assunto para outro artigo.

Uma vez que existem duas variáveis declaradas na classe TFormPrincipal, elas precisaram ser criadas em algum momento. É aí que entra o evento OnCreate do formulário, pois quando o formulário for criado também será criado um objeto da classe TVenda e outro da classe TBalanco. Veja isso entre as linhas 11 e 15.

Entre as linhas 26 e 30 está o evento OnClick do botão BotaoIniciarBalanco. Observe que nada de especial acontece neste método. O objeto Balanco criado no evento OnCreate – linha 14 – é usado para invocar o método IniciarBalanco e em seguida o Caption do TLabel LabelStatusBalanco é alterado para indicar que um balanço está em andamento. A mesma coisa é feita para os demais botões deste formulário. Todos os botões apenas invocam o método que vai executar a ação que eles se propõem a executar e em seguida mudam o status do TLabel, indicando que a ação foi executada. Assim com o evento OnClick do botão BotaoIniciarBalanco.

Listagem 3. Implementação da classe TFormPrincipal

```
1   private
2    Venda: TVenda;
3    Balanco: TBalanco;
4   public
5    { Public declarations }
6   end;

7   var
8   FormPrincipal: TFormPrincipal;

9   implementation

10  {$R *.dfm}

11  procedure TFormPrincipal.FormCreate(Sender: TObject);
12  begin
13   Venda := TVenda.Create;
14   Balanco := TBalanco.Create;
15  end;

16  procedure TFormPrincipal.BotaoFinalizarBalancoClick(Sender: TObject);
17  begin
18   Balanco.FinalizarBalanco;
19   LabelStatusBalanco.Caption := 'Nenhum balanço inciado';
20  end;

21  procedure TFormPrincipal.BotaoFinalizarVendaClick(Sender: TObject);
22  begin
23   Venda.FinalizarVenda;
24   LabelStatusVenda.Caption := 'Nenhuma venda inciada';
25  end;

26  procedure TFormPrincipal.BotaoIniciarBalancoClick(Sender: TObject);
27  begin
28   Balanco.IniciarBalanco;
29   LabelStatusBalanco.Caption := 'Balanço em adamento';
30  end;

31  procedure TFormPrincipal.BotaoIniciarVendaClick(Sender: TObject);
32  begin
33   Venda.IniciarVenda;
34   LabelStatusVenda.Caption := 'Venda em adamento';
35  end;
```

Não posso deixar passar despercebido o fato de que as classes TVenda e TBalanco não se conhecem, pois em nenhum momento uma está fazendo referência para a outra. Preste muita atenção neste detalhe, pois mesmo sem ter conhecimento uma da outra, ou seja, a classe TBalanco vai avisar a classe TVenda quando um balanço for iniciado ou finalizado.

Implementado o padrão Observer

Padrão Observer

Define uma dependência um-para-muitos entre objetos de modo que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente.

Até agora tudo que fiz foi criar duas classes, cada uma com dois métodos. Além disto criei um formulário que usa os métodos destas classes. Nada de especial nisto. Da maneira como está, se você quiser integrar a classe TVenda com a classe TBalanco para que uma funcione de acordo com o que está acontecendo na outra, será necessário usar a técnica do TTimer discutido no início do artigo. Mas, se você não vai usar o TTimer, então como implementar o Padrão Observer? Como ele é ou funciona? Veja na **Figura 3** o diagrama de classes.

Este é o grupo de classes que deve existir para implementar o Padrão Observer. Veja que existem duas interfaces, são elas:

IObservavel e IObservador. É graças a elas que a classe TBalanco conseguirá enviar notificações para a classe TVenda sem saber que a classe TVenda existe. Além de duas interfaces existem também duas classes. Cada uma delas implementa uma das interfaces existentes.

Uma dica para que você entenda o diagrama de qualquer padrão, é que você interprete cada classe do diagrama como sendo um personagem do problema que pretende resolver. Por exemplo, na aplicação deste artigo existem duas classes que precisam se comunicar sem se conhecer: TVenda e TBalanco. Este é o problema existente. Após identificar o problema, basta ler e entender a descrição oficial do padrão para ver se ele resolve o problema. Neste caso, a descrição diz que o Padrão Observer permite que um objeto interessado em alguma informação se registre com o objeto que detém a informação, a fim de ser notificado quando a informação sofrer alterações. Esta informação pode ser um status que muda um evento que ocorre ou qualquer outra coisa. Em outras palavras, quando você tem um objeto que se comporta de maneira diferente de acordo com o que acontece em outro objeto, você pode usar o Padrão Observer, pois assim é possível fazer estes dois objetos se comunicarem através de uma interface comum sem que um saiba da existência do outro.

Com base neste entendimento, é possível perceber que existem dois tipos de objetos envolvidos no problema que o Padrão Observer pretende resolver: um objeto interessado em alguma informação e outro objeto que detém a informação. Este é exatamente o caso da aplicação de exemplo deste artigo. Existe a classe TVenda que está interessada em uma informação da classe TBalanco. Eu digo que TVenda está interessada em algo da classe TBalanco, pois de acordo com o que acontece na classe TBalanco a classe TVenda terá um comportamento diferente. A regra é a seguinte: não pode existir uma venda em andamento enquanto o balanço estiver em andamento. Se alguma venda estiver em andamento quando o balanço começar, então a venda deve ser cancelada. Em comparação com o diagrama da **Figura 3**, a classe TVenda seria a classe representada por TObservadorConcreto, enquanto que a classe TBalanco está representada por TObservavelConcreto.

As interfaces envolvidas

Princípio de orientação a objetos

Programe para uma interface, não para uma implementação.

Repare bem nos nomes das duas interfaces do diagrama apresentado na **Figura 3**. Eles são nomes bem sugestivos: IObservavel e IObservador. Como o próprio nome sugere, a interface IObservador deve ser implementada pelos objetos que desejam observar outros objetos, ou seja, desejam ser notificados quando algo acontecer em outro objeto. Este é o caso da classe TVenda que vai observar a classe TBalanco. Por outro lado existe a interface IObservavel, que como o nome sugere é a interface implementada por objetos que serão observados. Este é o caso da classe TBalanco que será observada pela classe TVenda.

Invertendo seu raciocínio

Quando eu digo que TVenda vai observar TBalanco tem-se a impressão de que a classe TVenda fará solicitações constantes para a classe TBalanco para saber se algo mudou. Mas não é assim que funciona – isto seria a solução do TTimer. Com o Padrão Observer a classe que está observando não verifica se algo mudou, ao invés disto, ela é avisada quando algo muda. Isto é possível, pois a classe observadora (TVenda) se registra com o objeto que detém a informação (TBalanco), e quando algo mudar neste objeto (TBalanco), ele irá avisar todas as classes registradas (neste caso TVenda) que a informação mudou. Ou seja, a classe observada avisa quem está observando que uma mudança ocorreu, ao invés da classe que está observando perguntar insistentemente se algo mudou.

A inversão de raciocínio está no seguinte fato: a classe que está sendo observada irá notificar as classes que estão observando ao invés das classes que estão observando ficarem constantemente verificando se algo mudou. É mais ou menos assim: "Não fique me ligando para saber se tem vaga. Quando surgir uma vaga, eu ligarei!". É daí que surge a expressão "Não ligue para mim. Eu Ligo para você!", comumente usada para explicar o Padrão Observer. Este título diz respeito à relação entre as classes observadoras e observáveis.

Como uma classe observa a outra

A interface IObservavel possui três métodos: um para registrar um objeto observador, outro para cancelar o registro e outro para notificar todos os objetos registrados. Isto significa que para uma classe se tornar observadora de outra, basta se registrar através do método RegistrarObservador. Desta maneira a classe que está sendo observada guarda uma referência para a classe que está observando. Quando algo acontecer todas as classes que se registraram, serão notificadas.

Uma classe que deseja observar outra, precisa obrigatoriamente implementar a interface IObservador. As classes que serão observadas precisam implementar a interface IObservavel. Isto significa que todas as classes observadoras terão o método Notificacao, como você pode ver na interface IObservador exibida na **Figura 3**. É através deste método que as classes observadoras serão avisadas quando algo mudar nas classes que estão sendo observadas.

Para o exemplo deste artigo isto significa que a classe TVenda vai se registrar com a classe TBalanco para saber quando um balanço for iniciado ou finalizado. Mais tarde quando o balanço for iniciado ou finalizado a classe TVenda será notificada através do método Notificacao que algo aconteceu na classe TBalanco. Veja como tudo isto funciona.

Implementando as interfaces

Antes de começar a criar e implementar as interfaces do Observer é necessário criar as classes TBalanco e TVenda. Se você ainda não criou, adicione uma nova unit através menu File > New > Unit. Salve esta unit com o nome Venda.pas e digite o código da **Listagem 1**. Adicione uma nova unit e salve com o nome Balanco.pas. Desta vez copie o código da **Listagem 2**. Para finalizar implemente as ações dos botões OnClick de cada botão do formulário. As instruções executadas no evento destes

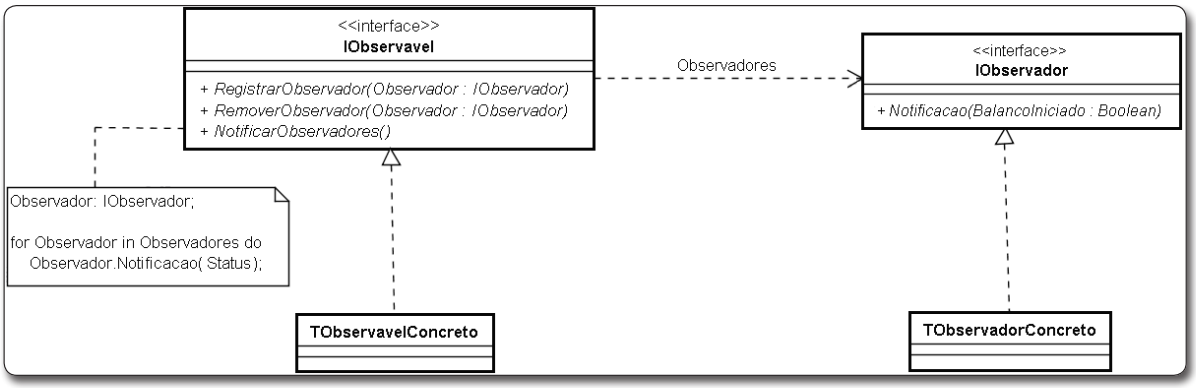


Figura 3. Diagrama do padrão Observer

botões estão na **Listagem 3**. Não se esqueça de transcrever as instruções do evento OnCreate (FormCreate). Elas também aparecem na **Listagem 3**. Além disto, para que o projeto compile corretamente é necessário declarar as units Balanco.pas e Venda.pas na unit Principal.pas, pois como você vê, o formulário principal usa os objetos destas units.

Agora compile e execute o projeto. Se você fez tudo certo o programa vai abrir sem mais transtornos. Com o programa em execução, inicie uma venda clicando no botão Iniciar venda. Ao fazer isto uma venda é iniciada, com isto o objeto da classe TVenda mostra um ShowMessage avisando que a venda iniciou. Faça a mesma coisa para o balanço. Inicie um balanço clicando no botão Iniciar balanço. O balanço será iniciado e o objeto da classe TBalanco também exibirá uma mensagem dizendo que o balanço iniciou.

Aparentemente tudo certo, porém, você sabe que não pode existir uma venda e um balanço iniciado ao mesmo tempo. Para corrigir isto vamos implementar o Padrão Observer para que a classe TVenda seja capaz de tomar as devidas providências quando um balanço começar. Crie uma nova unit e salve-a com o nome Observer.pas. Nesta nova unit crie as interfaces IObservador e IObservavel. Veja **Listagem 4**.

Na **Listagem 4** você vê a declaração das duas interfaces envolvidas no padrão. Entre as linhas 5 e 8 está a declaração da interface IObservador. Tenha em mente que esta interface deve ser implementada pelos objetos que desejam observar outro objeto. Repare o nome da interface: IObservador, ou seja, é o objeto observador – aquele que observa outro. Este é o caso da classe TVenda que deseja observar a classe TBalanco. Todos os objetos que implementarem esta interface passam a ser considerados um objeto observador e precisam obrigatoriamente implementar o método Notificacao, que está declarado na interface – veja linha 7. Isto significa que a classe TVenda vai possuir o método Notificacao, que será invocado pela classe TBalanco quando necessário. Tudo isto vai acontecer sem que uma classe precise conhecer a outra. Elas vão conhecer apenas as interfaces envolvidas.

Entre as linhas 9 e 14 está a declaração da interface IObservavel. Todos os objetos que permitem ser observados por outros objetos devem implementar esta interface. Assim como IObservador, o nome desta classe também é uma sugestão, repare: IObservavel, ou seja, que está sendo observado – é o caso da classe TBalanco que será observada pela classe TVenda. Esta interface possui três métodos, porém, somente dois deles são invocados por outros objetos, são eles: RegistrarObservador e RemoverObservador. Já o método NotificarObservadores é normalmente usado pela própria classe que implementa a interface, embora ele seja

Listagem 4. As interfaces IObservador e IObservavel

```
1  unit Observer;
2  interface
3  type
4    IObservavel = interface;//declaração antecipada
5    IObservador = interface
6      ['{506387BD-E9AA-4419-8E1A-C5A26EBEC7A9}']
7      procedure Notificacao(BalancoIniciado: Boolean);
8    end;
9    IObservavel = interface
10     ['{88684F1C-9432-4F8D-AF70-0365334AFA52}']
11     procedure RegistrarObservador(Observador: IObservador);
12     procedure RemoverObservador(Observador: IObservador);
13     procedure NotificarObservadores;
14   end;
15 implementation
16 end.
```

Nota do DevMan

Se você não tem muita experiência com as interfaces da linguagem Delphi, deve ter estranhado o código que aparece nas linhas 6 e 10 da **Listagem 4**. Este código é chamado de GUID - Globally Unique Identifier e serve para tornar sua interface única. Tudo o que você precisa saber sobre a GUID é que sua declaração não é obrigatória para criar uma interface, porém, você ganha uma série de facilidades ao declará-la. Fora as facilidades com type cast, você também ganha a capacidade de testar se uma classe implementa ou não uma interface, como no código **if Supports(Obj, INomeDaInterface) then**. Se você não declarar o GUID em suas interfaces, o teste com Supports não funciona. A IDE pode gerar um GUID aleatório para você, basta pressionar Ctrl + Shift + G.

Nota

É comum que cada unit do projeto tenha a declaração de apenas uma classe ou unit, mas neste caso eu fui obrigado pelo compilador a declarar as duas interfaces no mesmo arquivo, porque como uma interface usa a outra, o compilador reclama de referência circular entre as units quando elas estão em units separadas.

declarado na sessão public. Nada impede que outro objeto possa invocar este método, porém esta é uma situação mais difícil de acontecer. Se precisar invocá-lo, não evite.

Os métodos mais importantes são os de registro e cancelamento de registro. Quando uma classe quiser se tornar observadora de outra ela deve se registrar para receber as notificações: isto será feito através do método RegistrarObservador. E para não receber mais notificações, basta cancelar o registro invocando o método RemoverObservador. Veja como tudo isto funciona. A **Listagem 5** mostra a classe TBalanco implementando a interface IObservavel. Altere a classe TBalanco que você criou para que seja igual ao código da **Listagem 5**.

A primeira diferença notável é que TBalanco passa a ser uma classe que estende TInterfacedObject ao invés de TObject e implementa IObservavel. Para isto acontecer foi necessário declarar a unit Observer.pas – criada na **Listagem 4** – na sessão uses: veja linha 3. Nas linhas 10 e 11 estão as declarações dos métodos IniciarBalanco e FinalizarBalanco. Estes métodos você já conheceu na **Listagem 2**, por isto a implementação deles não aparece novamente nesta listagem. Em seguida você tem quatro novos métodos, sendo três deles obrigatórios por causa da implementação da interface. Eu digo obrigatório, pois se você tirar um deles será gerado erro de compilação. Uma vez que a classe TBalanco implementa a interface IObservavel, ela deve implementar todos os métodos da interface.

Entre as linhas 19 e 23 está a implementação do método RegistrarObservador. O método possui um parâmetro Observador do tipo da interface IObservador, ou seja, pode ser qualquer objeto, desde que este implemente a interface IObservador. Internamente a classe TBalanco guarda uma referência ao objeto que está se registrando dentro da variável Observadores – veja na linha 22. A variável Observadores está declarada na linha 7 e é criada no construtor da classe, veja linha 42. Mais tarde como você verá, todos os objetos que estiverem dentro da variável Observadores serão notificados das mudanças que ocorrem em TBalanco.

Entre as linhas 24 e 31 está a implementação do método RemoverObservador usado para cancelar o registro de um objeto que não deseja mais receber notificações sobre as mudanças que ocorrem em TBalanco. Este método simplesmente remove o observador – que está cancelando o registro – da variável Observadores. A variável Observadores é uma variável do tipo TList. O método TrimExcess da classe TList, invocado na linha 29 simplesmente ajusta a quantidade de elementos existentes da lista. Quando TrimExcess é invocado todas as posições da lista que não apontam para objeto algum são removidas da lista. Para mais informações sobre este método você deve consultar a documentação da classe TList.

Uma vez que existam objetos registrados será necessário notificá-los quando algo acontecer. Para isto serve o método NotificarObservadores que pode ser visto entre as linhas 32 e 38. Este método simplesmente itera com todos os observadores registrados, que estão na variável Observadores e para cada observador registrado o método Notificacao é invocado.

Listagem 5. TBalanco implementa IObservavel

```
1  unit Balanco;

2  interface

3  uses Generics.Collections, Observer;

4  type
5   TBalanco = class(TInterfacedObject, IObservavel)
6   private
7    Observadores: TList<IObservador>;
8    BalancoIniciado: Boolean;
9   public
10    procedure IniciarBalanco;
11    procedure FinalizarBalanco;
12    procedure RegistrarObservador(Observador: IObservador);
13    procedure RemoverObservador(Observador: IObservador);
14    procedure NotificarObservadores;
15    constructor Create;
16 end;

17 implementation

18 { Implementação dos método IniciarBalano e FinalizarBalanco, aqui }

19 procedure TBalanco.RegistrarObservador(Observador: IObservador);
20 begin
21  if not Observadores.Contains(Observador) then
22   Observadores.Add(Observador);
23 end;

24 procedure TBalanco.RemoverObservador(Observador: IObservador);
25 begin
26  if Observadores.Contains(Observador) then
27  begin
28   Observadores.Remove(Observador);
29   Observadores.TrimExcess;
30  end;
31 end;

32 procedure TBalanco.NotificarObservadores;
33 var
34  Observador: IObservador;
35 begin
36  for Observador in Observadores do
37   Observador.Notificacao(BalancoIniciado);
38 end;

39 constructor TBalanco.Create;
40 begin
41  inherited Create;
42  Observadores := TList<IObservador>.Create;
43 end;

44 end.
```

O segredo por trás do padrão Observer

Repare que em nenhum momento a classe TBalanco fez referência à classe TVenda ou qualquer outra que seja. Em todos os momentos esta classe trabalhará apenas com as interfaces IObservador e IObservavel. Na linha 37, quando a classe TBalanco invoca o método Notificacao para avisar aos observadores registrados que algo aconteceu, ela – a classe TBalanco – não sabe com qual objeto está se comunicando. Nós sabemos que será a classe TVenda neste exemplo, porém de fato, poderia ser qualquer uma pois a classe TBalanco não se preocupa com o objeto a ser notificado, uma vez que ela chama o método Notificacao da interface. E quem é que vai implementar o método Notificacao?

Qualquer objeto que desejar se registrar com a classe TBalanco para receber notificações deve implementar a interface IObservador. Eu sei disso, pois o método RegistrarObservador da classe TBalanco recebe um objeto qualquer desde que ele implemente a interface esperada. Veja na linha 12 da **Listagem 6**. Veja agora

como a classe TVenda implementa a interface IObservador e consequentemente o método Notificacao.

Criando uma classe observadora

A classe TVenda deseja receber notificações sobre o que acontece com a classe TBalanco. Para isto ela precisa implementar a interface IObservador para poder fazer o registro de observadora com a classe TBalanco. Veja na **Listagem 6** como fica a classe TVenda após implementar a interface necessária. Altere a classe TVenda que você criou para que fique o código apresentado.

A regra é clara: ao implementar uma interface a classe deve escrever o corpo de todos os métodos que a interface declara. Sendo assim, a classe TVenda precisou escrever o corpo do método Notificacao declarado na interface IObservador, pois, como você pode ver na linha 5 da **Listagem 6** a classe TVenda está implementando a interface IObservador. É por este motivo que a unit Observer.pas foi declarada na sessão uses – veja linha 3. A declaração dos três métodos existentes na classe pode ser vista entre as linhas 10 e 12. Os métodos IniciarVenda e FinalizarVenda você já conheceu na **Listagem 1**. Já o método Notificacao é implementado entre as linhas 16 e 25 da **Listagem 6**.

Listagem 6. TVenda implementa IObservador

```
1  unit Venda;

2  interface

3  uses Observer;

4  type
5   TVenda = class(TInterfacedObject, IObservador)
6   private
7    VendaIniciada: Boolean;
8    BalancoIniciado: Boolean;
9   public
10     procedure IniciarVenda;
11     procedure FinalizarVenda;
12     procedure Notificacao(BalancoIniciado: Boolean);
13  end;

14  implementation

15  { Implementação do método FinalizarVenda, aqui }

16  procedure TVenda.Notificacao(BalancoIniciado: Boolean);
17  begin
18   if VendaIniciada and BalancoIniciado then
19   begin
20    ShowMessage('Desculpe! A venda será cancela porque um outro ' +
21     'usuário iniciou o balanco.');
22    VendaIniciada := False;
23   end;
24   Self.BalancoIniciado := BalancoIniciado;
25  end;

26  procedure TVenda.IniciarVenda;
27  begin
28   if not VendaIniciada and not BalancoIniciado then
29   begin
30    VendaIniciada := True;
31    ShowMessage('Venda iniciada!');
32   end;
33  end;

34  end.
```

Como disse, o método Notificacao será invocado pela classe TBalanco quando o balanço for iniciado ou finalizado. Este método tem um parâmetro do tipo Boolean indicando se o balanço está em andamento ou não. Sabendo disto, na linha 18, a classe TVenda verifica se existe uma venda em andamento e se o balanço foi iniciado. Caso isto seja verdade, uma mensagem é apresentada para o usuário e em seguida na linha 22 a venda é cancelada. Na linha 24 armazena-se o status atual do balanço, pois se existir um balanço iniciado, nenhuma venda poderá ser iniciada. Você nota isto na nova implementação do método IniciarVenda. Veja na linha 28. A única mudança no método IniciarVenda é que agora a venda só será iniciada caso não exista um balanço iniciado. A variável BalancoIniciado é do tipo Boolean e está declarada na linha 8.

Registrando classes observadoras

Agora tenho em mãos duas classes prontas para se comunicar sem se conhecerem, graças ao uso das interfaces. Mas como o registro será feito? Uma solução bem interessante é fazer o registro com a classe TBalanco quando a classe TVenda é criada. Então terei que implementar o construtor da classe TVenda para que o registro aconteça. Veja o código a seguir:

```
1  constructor TVenda.Create;
2  begin
3   inherited;
4   TBalanco.Create.RegistrarObservador(Self);
5  end;
```

Parece ser uma boa solução, mas não é. Ao fazer isto estou declarando que a classe TVenda tem conhecimento da classe TBalanco e este não é objetivo. Além disso, este código não vai funcionar. Ele não gera erro ou coisa parecida, porém, a classe TVenda não receberá as notificações, pois como você pode ver na linha 4 a classe TVenda está criando um novo objeto de TBalanco para se registrar, enquanto que a classe TFormPrincipal usa outro objeto de balanço. Veja na linha 14 da **Listagem 3** que outro objeto de balanço está sendo criado pelo formulário, ou seja, esta solução falha porque a classe TVenda se registra com o objeto Y da classe TBalanco enquanto que o objeto X da classe TBalanco é quem vai iniciar ou finalizar o balanço. Isto não aconteceria com o Padrão Singleton – mas não estou utilizando ele e mesmo que estivesse não quero que a classe TVenda conheça a classe TBalanco. Se fosse assim eu usaria o TTimer.

Quem vai fazer este registro então é a própria classe TFormPrincipal. A classe TFormPrincipal conhece as duas classes como você pode ver na **Listagem 3**. Então, assim que o formulário cria os objetos destas classes ela mesma faz o registro de uma com a outra. Veja na **Listagem 7**. Mude a implementação do evento OnCreate do formulário para que fique de acordo com o apresentado.

Listagem 7. TFormPrincipal faz o registro de TVenda

```
1  procedure TFormPrincipal.FormCreate(Sender: TObject);
2  begin
3   Venda := TVenda.Create;
4   Balanco := TBalanco.Create;
5   Balanco.RegistrarObservador(Venda);
6  end;
```

A única diferença entre o evento FormCreate apresentado na **Listagem 7** e o evento OnCreate apresentado na **Listagem 3** é a linha 5. Nesta linha é feito o registro do objeto Venda com o

objeto Balanco. A partir de agora o objeto Venda receberá notificações sempre que um balanço for iniciado ou finalizado. Na verdade, ainda falta um detalhe muito importante. Você notou que apesar de implementar o método NotificarObservadores na classe TBalanco (linha 32 da **Listagem 5**), ele não foi invocado em nenhum lugar? Ou seja, os observadores registrados nunca serão notificados sobre as mudanças.

A ideia é que os observadores sejam notificados sempre que um balanço for iniciado ou finalizado, portanto o método NotificarObservadores deve ser invocado a partir dos métodos IniciarBalanco e FinalizarBalanco. Veja na **Listagem 8** como fica.

Listagem 8. Notificando os observadores

```
1  procedure TBalanco.IniciarBalanco;
2  begin
3    if not BalancoIniciado then
4    begin
5      BalancoIniciado := True;
6      NotificarObservadores;
7      ShowMessage('Balanco Iniciado!');
8    end;
9  end;
10
11 procedure TBalanco.FinalizarBalanco;
12 begin
13   if BalancoIniciado then
14   begin
15     BalancoIniciado := False;
16     NotificarObservadores;
17     ShowMessage('Balanco finalizado!');
18   end;
```

Estes métodos continuam iguais aos que foram apresentados para você na **Listagem 2**, exceto pelo fato de que agora, antes de mostrar uma mensagem confirmando que ação foi efetuada, todos os observadores são notificados de que o status do balanço mudou. Agora você pode compilar e executar o exemplo. Com a aplicação em execução, inicie um balanço e em seguida tente iniciar uma venda. Você verá que nada vai acontecer, pois quando você iniciou o balanço a venda foi notificada disto e ela só irá iniciar uma venda quando o balanço terminar.

Agora finalize o balanço, inicie uma venda e em seguida inicie o balanço novamente. Uma mensagem dizendo que a venda em aberto será fechada, pois o balanço foi iniciado será mostrada. Esta mensagem é criada pela classe TVenda, como você pode ver na linha 20 da **Listagem 6**. Em seguida uma mensagem dizendo que o balanço iniciou é exibida. Esta segunda mensagem é provida da classe TBalanco.

Nota

Mude a implementação dos métodos IniciarBalanco e FinalizarBalanco, que você copiou na **Listagem 2**, para que fique de acordo com o apresentado na **Listagem 8**.

Conclusão

Com o Padrão Observer você consegue desacoplar as partes do sistema uma das outras e mesmo assim permite que elas interajam entre si, possibilitando que você crie um sistema integrado de verdade sem a necessidade de espalhar TTimers por toda aplicação.

Todo o segredo deste padrão está no uso das interfaces. Seguindo o princípio de sempre programar para interfaces foi possível fazer dois objetos trocarem informações entre si sem a necessidade de se conhecerem. Existe ainda outra versão do Padrão Observer. Neste artigo eu mostrei a versão conhecida como "Modo Empurrar", pois a classe TBalanco está empurrando a informação para as classes observadoras. Veja a assinatura do método Notificacao da interface IObservador, que querendo ou não o observador vai receber o status atual do balanço e somente ele, porém se a classe observadora estiver interessada em alguma outra informação do balanço, ela não terá.

A outra forma seria versão conhecida como "Método Puxar", e no lugar do parâmetro "BalancoIniciado" do método Notificacao, existiria um parâmetro que poderia se chamar "QuemNotificou" do tipo IObservavel. Assim a classe que está sendo notificada teria uma referência ao objeto que fez a notificação e assim poderia obter qualquer informação que ela desejar.

Para você se tornar expert no Padrão Observer eu recomendo que corrija o bug existente na classe TFormPrincipal da aplicação de exemplo. Se você iniciar uma venda, o TLabel indica que a venda foi iniciada. Em seguida você inicia um balanço e a venda é cancelada automaticamente pela classe TVenda, porém o formulário não toma conhecimento disto, e o TLabel continua dizendo que a venda está iniciada. Uma solução para corrigir este problema é tornar a classe a TFormPrincipal observadora de TVenda e assim mudar o status do TLabel conforme o status da venda for mudando. É claro que para a classe TVenda notificar a classe TFormPrincipal de que algo mudou, a TVenda deve implementar IObservavel e a TFormPrincipal deve implementar IObservador. Neste caso TVenda seria tanto uma classe observadora como uma classe observável, pois implementaria as duas interfaces ao mesmo tempo. Talvez neste caso fosse até mais interessante usar o método Puxar do Padrão Observer.

Agora é com você. Boa diversão em seus estudos! ●

Links

Interfaces in Delphi Programming
http://delphi.about.com/od/oopindelphi/a/interfaces-in-delphi-programming-101.htm

GoF Patterns – Quick Review
http://www.mydeveloperconnection.com/html/gof_design_patterns.htm

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Delphi Prism e ASP.NET

Reaproveite seus conhecimentos Win32 / Desktop para criar aplicações Web – Parte 2

Guinther Pauli
guinther@devmedia.com.br
guinther.pauli@gmail.com
http://twitter.com/guintherpauli
http://gpauli.com
Desenvolve em Delphi há 15 anos, tendo utilizado a ferramenta desde a versão 1, passando pelo Turbo Pascal e Borland Pascal for Windows. Certificado em Delphi 3, 5, 6, 7, 2005, 2006, Delphi para Web e Delphi para Linux, palestante em 5 edições da Borland Conference. Certificado Microsoft MCP, MCAD, MCSD.NET, MCTS e MCPD (C#, ASP.NET), utilizando a plataforma .NET desde a versão beta. Editor das Revistas ClubeDelphi, .NET Magazine Brasil e EASY .NET Magazine, professor de Pós-Graduação da UNIVEM-SP, na área .NET. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, Visual Studio, atuando também em projetos de migração orientando empresas e desenvolvedores na adoção de melhores práticas, ministra treinamentos em .NET focados em equipes Delphi. Foi arquiteto e desenvolvedor ASP.NET do site de doações on-line do projeto Criança Esperança da Rede Globo / Unesco.

Do que trata o artigo
Lançado pela Embarcadero, o Delphi Prism é a nova forma de desenvolver aplicações para o .NET Framework e Mono utilizando o Object Pascal e o IDE do Visual Studio. Nesta série de artigos, veremos como utilizar o Delphi Prism para criar um site de E-Commerce usando a tecnologia ASP.NET.

Para que serve
O objetivo principal desta série de artigos é familiarizar os desenvolvedores Delphi Win32 com o ambiente de desenvolvimento do Delphi Prism, o IDE do Visual Studio, bem como introduzir o desenvolvimento de aplicações Web. Dessa forma, faremos vários comparativos entre os dois IDEs e também entre os Frameworks ASP.NET e VCL, servindo como um excelente guia para diminuir a curva de aprendizado ao se aprender ASP.NET baseando-se nos seus prévios conhecimentos em Delphi Win32 (Desktop).

Em que situação o tema é útil
Esta série será muito útil para quem nunca desenvolveu aplicativo algum para Web, nem mesmo teve um primeiro contato com o Delphi Prism / Visual Studio. Ao desenvolver aplicações para Web, ganhamos inúmeras vantagens se compararmos ao ambiente Desktop: deploy simplificado, uso de computação nas nuvens, não há necessidade de instalações visto que os usuários acessam um mesmo servidor na internet, não há dependência do sistema operacional pois usamos um browser, não há preocupação com backups nos clientes, nem altos custos para manutenções e visitas aos clientes etc.

Resumo do DevMan

No artigo da edição passada, tivemos uma introdução ao Delphi Prism, seus objetivos e propósitos. A seguir partimos para a criação do banco de dados no SQL Server que será usado ao longo desta série. Neste artigo vamos iniciar a criação do Web Site, aproveitando para falar dos fundamentos do ASP.NET e Visual Studio, sempre comparando-os com a VCL e RAD Studio, com o intuito de facilitar o aprendizado. Veremos a estrutura básica de um Web Site e iniciaremos a construção do layout básico do site de E-Commerce (que se chamará "Computer Store"), bem como o sistema de navegação.

No Visual Studio (chamado a partir de agora simplesmente de VS), clique em File > New Web Site. Na janela que aparece, que é semelhante ao Object Repository do IDE do Delphi Win32, selecione o template ASP.NET Web Site. Em Language escolha Oxygene (Delphi Prism) e em Location escolha File System, dando um diretório para a nossa aplicação, nesse caso defini C:\LOJA. Todas essas opções podem ser vistas na **Figura 1**.

Repare alguns detalhes interessantes nesta janela. Primeiro, veja que é possível escolher a versão do Framework que você quer utilizar no corrente projeto. Como estamos usando o Delphi Prism com VS 2008, escolhemos o .NET Framework 3.5. Desde o VS 2005 é possível que o desenvolvedor utilize uma nova versão do IDE, por exemplo, usando uma versão diferente do framework. Traduzindo para o Delphi tradicional, seria como se você pudesse usar o IDE do Delphi 2010, porém informando que desejaria usar a VCL na versão que acompanha o Delphi 7, ou ainda a CLX. Isso é útil em várias situações, como por exemplo, para manter retro compatibilidade com versões antigas do IDE. Seria possível abrir este projeto no VS 2005, por exemplo, caso fosse definido que o projeto utilizaria .NET Framework 2.0. Veja nesse caso, obviamente, que novos controles que surgiram no ASP.NET 3.5 não estariam presentes. Vale lembrar que o .NET Framework 3.0 não teve um IDE do VS correspondente quando do seu lançamento.

Dica

Certamente quando você estiver lendo este artigo, o Delphi Prism para o VS 2010 (ainda em beta) já estará lançado e você poderá criar este Web Site já em ASP.NET 4.0. No momento que escrevo este artigo, o VS 2010 já tinha sido lançado e uma versão do Prism está sendo aguardada para suporte ao .NET 4.0. Já a partir do próximo artigo, se tudo ocorrer conforme os planos da Embarcadero, que previu o Prism para VS 2010 para final de Maio, nós já vamos continuar nosso curso no novo IDE da Microsoft, usufruindo de seus novos recursos.

Uma outra particularidade é a opção Location. Até o VS 2003 (.NET Framework 1.1), era necessário utilizar o servidor IIS (Internet Information Services) da Microsoft durante o desenvolvimento da aplicação. Isso envolvia, além de ter a ferramenta instalada, ter configurado e registrado o .NET Framework no servidor Web, criar um diretório virtual para rodar a aplicação, além de ser bastante trabalhoso debugar uma aplicação rodando nele. Além disso, se você, por exemplo, levasse uma aplicação ASP.NET para abrir em outra máquina, precisava configurar o servidor e diretório virtual, e algumas vezes até permissões. Essa opção ainda existe, caso queira iniciar o desenvolvimento já usando o IIS, ela se chama http (na combo de opções). Também existe a FTP, no caso do IIS estar em uma máquina remota. Mas a que usaremos e que com certeza você utilizará sempre é a File System, que apenas solicita o nome de um diretório. Nada de IIS (durante o desenvolvimento), nem configuração, nem diretórios virtuais, nem permissões. Para abrir o site novamente, basta dar Open Web Site e localizar o diretório. Para rodar, basta abrir e dar um F5 ou Ctrl+F5. Não existe um arquivo de projeto nesse caso, tudo que está no diretório é considerado parte do Web Site. É como o Delphi Win32, basta abrir um DPR, F9 e rodar, nada de configuração. Isso se dá graças a um servidor local de desenvolvimento que fica rodando e escutando a porta 80, atendendo a requisições HTTP. É um servidor local somente para desenvolvimento ASP.NET, presente desde o VS 2005. Facilita os testes, depuração, como o Web App Debugger para quem já desenvolveu com Delphi Win32 e WebBroker / WebSnap, ou o Cassini Web Server para quem desenvolveu em ASP.NET com Delphi for .NET (BDS).

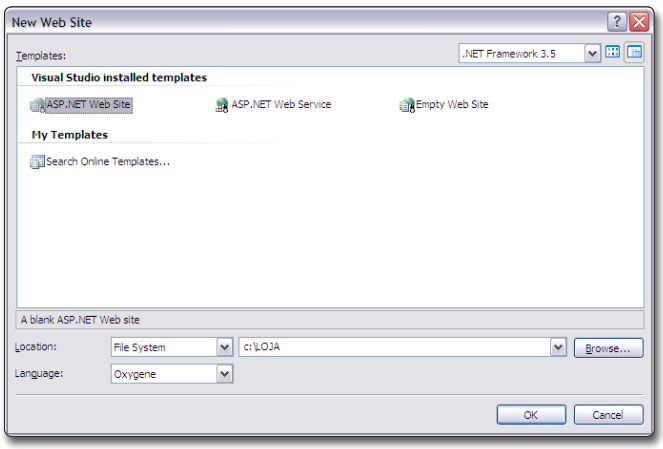


Figura 1. Criando o ASP.NET Web Site no Delphi Prism

Estrutura padrão do Web Site

Criado o projeto, vamos analisar a estrutura básica criada, como mostra a Solution Explorer (**Figura 2**). Veja que o projeto é o próprio diretório, como comentei. Temos também uma pasta especial App_Data, especial do ASP.NET, podemos colocar nela arquivos de banco de dados do SQL Server, arquivos XML, ou qualquer outro tipo de informação que precise estar segura e que não possa ser acessada pela Web. Por exemplo, não seria nada interessante um usuário abrir o browser e digitar algo como "http://www.meudominio.com/LOJA/LOJA.mdf" e conseguir baixar todo o BD do Web Site. Isso não ocorre porque o ASP.NET protege a pasta App_Data. Além disso, vale lembrar que nosso BD não reside no mesmo diretório do Web Site (apesar de possível), ele foi criado no SQL Server Management Studio Express e está registrado e visível na nossa aplicação. Por questões de segurança, é altamente recomendável que ele esteja em outra máquina da intranet visível somente ao Web Server.

Web.Config, como o próprio nome sugere, é o arquivo de configuração da aplicação, em formato XML, claro. Aqui podemos configurar uma série de itens para a nossa aplicação Web que são oferecidos pelo ASP.NET, desde o comportamento do uso de memória do processo servidor, número de requisições que podem ficar em espera, tamanho máximo das requisições, duração das sessões, modos de autenticação etc. Veremos algumas dessas configurações ao longo deste curso. Uma que com certeza você vai gostar de saber desde já é que a string de conexão com o BD fica nesse arquivo,

o que facilita o deploy, caso você coloque o banco em um servidor com nome diferente (o normal). Algo como faz o dbxconnections.ini do dbExpress. Vale lembrar, obviamente, que esse arquivo não é compilado com a aplicação e deve ser distribuído à parte.

Default.aspx é a nossa página, é o nosso "TForm1". Note que ele é composto por dois arquivos, chamados respectivamente de Default.aspx e Default.aspx.pas. O arquivo ASPX define a interface, a estrutura gráfica de uma página. Indo direto ao ponto, ele é o arquivo "DFM" de um formulário em aplicações ASP.NET. Se colocássemos um botão nesse form, o ASPX teria a cara da **Listagem 1**. Repare que temos uma diretiva logo no início, chamada Page, indicando que a linguagem utilizada aqui será a Oxygene (Delphi Prism). Repare também dois elos de ligação com o nosso outro arquivo, o .PAS, através dos atributos CodeFile e Inherits. O ASPX, como comentei, tem papel semelhante ao DFM, tanto que podemos ver no código a declaração textual do botão de teste. Propositalmente, adicionei um manipulador para o evento Click do botão, para você ver que funciona da mesma forma que no Delphi Win32. A diferença aqui é que não há RTTI (Runtime Type Information) ou Reflexão, como chamamos a introspecção no universo .NET, o ASPX pode ser compilado com o form e não há necessidade da referência a esse botão em uma sessão published no .PAS.

Nota

Arquivos ASPX não dependem de linguagem. Isso quer dizer que se você construir essa mesma aplicação usando C#, VB.NET ou qualquer outra linguagem para o .NET, o código ASPX será o mesmo.

Examinando o código .PAS gerado (**Listagem 2**), vemos uma semelhança incrível com uma aplicação tradicional Delphi Win32. Aqui esse arquivo é também conhecido como code-behind, um recurso que sempre tornou o ASP.NET diferente e superior a outras tecnologias para Web. Sim, há uma separação entre layout e comportamento (como na VCL). Pode parecer natural, mas outras linguagens, como o próprio ASP clássico ou PHP clássico, mesclavam código HTML e código servidor (VB, por exemplo) em um único arquivo. Note que aqui temos a sessão uses (o using para quem usa C#), os namespaces, que são semelhantes às units da VCL / RTL (porém com um nível de abstração muito maior), temos a declaração do formulário (o que seria o equivalente ao TForm1 = class (TForm)), os métodos e a implementação. Repare mais a fundo que realmente não há a necessidade da definição dos membros da página abaixo da declaração da classe, como na VCL. Isso ocorre desde o .NET Framework 2.0 com a inclusão do recurso de classes parciais (partial class). O compilador por baixo dos panos pega o que você definiu no ASPX e junta com a classe de mesmo nome que está no PAS, gerando uma única classe. Daí o termo parcial, pois parte está definida no ASPX e parte no code-behind (compilada vira uma só). Na prática isso significa que você pode acessar os controles servidores de forma muito simples, por exemplo:

```
method _Default.Button1_Click(
      sender: System.Object; e: System.EventArgs);
begin
  // Button1 não está definido no .PAS
  Button1.Text := "Ola Mundo";
end;
```

Estes são os fundamentos sobre a estrutura de um projeto ASP.NET, que são essenciais para você iniciar os trabalhos. Falaremos mais sobre fundamentos durante o desenrolar do desenvolvimento da aplicação.

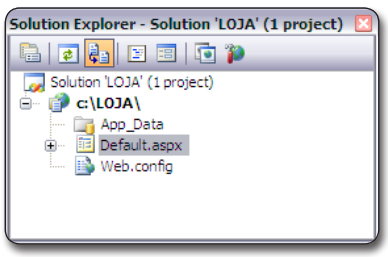


Figura 2. Estrutura padrão de um Web Site

Nota

No Solution Explorer, dê um clique de direita no default.aspx e apague-o do projeto (delete), pois não usaremos a página vazia (em branco), nós adicionaremos páginas vinculadas a uma Master Page.

Criando a Master Page

Aqui vai uma péssima notícia: o ASP.NET não suporta herança visual de formulários. Não aquela que estamos acostumados no Delphi Win32, uma herança que além de ser visual é também totalmente orientada a objetos, com suporte a polimorfismo e tudo mais. Porém existe um recurso que permite "simular" uma herança, pelo menos visual, através do uso de Master Pages, recurso introduzido no ASP.NET 2.0. Uma Master Page é como um formulário base do Delphi, nele você vai colocar todo o layout que é comum a todo o site, como cabeçalhos, menu de navegação, rodapés etc. Pode colocar também algum código padrão que será executado por todas as páginas. Porém não será possível acessar esse código nem seus itens a partir de outras páginas, pelo menos de uma forma direta como no Delphi Win32, onde o form é uma variável global (a opção é usar a propriedade Master de Page). Para criar a Master Page, dê um clique de direita no nome do projeto na Solution Explorer (ou seja, o nome do diretório) e escolha Add New Item (**Figura 3**). Fiz questão de colocar uma figura neste artigo para representar esta opção, pois pode ter certeza que você a utilizará muito a partir de agora. Escolha então a opção Master Page (**Figura 4**).

No formulário em branco que surgirá você verá um controle chamado ContentPlaceHolder. Tudo o que colocarmos **fora** deste controle será visto por todas as páginas. As páginas,

Listagem 1. Arquivo ASPX = DFM

```
<%@ Page Language="Oxygene" AutoEventWireup="true" CodeFile=
  "Default.aspx.pas" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>ClubeDelphi</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>

      <asp:Button ID="Button1" runat="server" onclick=
        "Button1_Click" Text="Button" />

   </div>
   </form>
</body>
</html>
```

Listagem 2. Arquivo .PAS = Code-Behind

```
namespace;

interface

uses
 System,
 System.Data,
 System.Configuration,
 System.Web,
 System.Web.Security,
 System.Web.UI,
 System.Web.UI.WebControls,
 System.Web.UI.WebControls.WebParts,
 System.Web.UI.HtmlControls;

type
 _Default = public partial class(System.Web.UI.Page)
 protected
   method Button1_Click(sender: System.Object; e: System.EventArgs);
   method Page_Load(sender: Object; e: EventArgs);
 end;

implementation

method _Default.Page_Load(sender: Object; e: EventArgs);
begin

end;

method _Default.Button1_Click(sender: System.Object; e: System.EventArgs);
begin

end;

end.
```

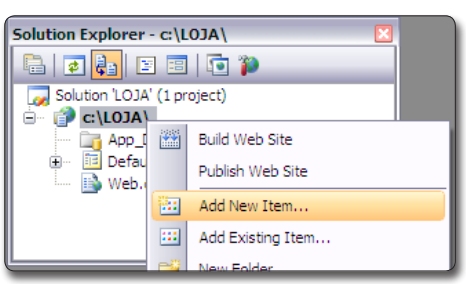


Figura 3. Adicionando um novo item

com os conteúdos específicos, serão "colocadas" dentro deste controle. Nossa Master será simples (você pode ver ela na **Figura 5**), vamos colocar uma imagem de topo, um controle Menu (guia Navigation), um elemento "Horizontal Rule" da guia HTML (o que gera uma tag HR do HTML), uma pequena imagem de logo no rodapé e uma mensagem de copyright (faça exatamente como no Delphi Win32, simplesmente arraste da ToolBox para o designer e configure as propriedades necessárias). Note que para a mensagem de rodapé você não precisa colocar um Label, pode digitar diretamente no form. Para colocar as imagens, dê um clique de direita no nome do projeto, escolha New Folder e dê a ele o nome de Images. A seguir, coloque imagens a sua escolha na pasta (usando o Windows Explorer), dê um refresh e arraste-as para o form. Para o menu, basta abrir a SmartTag do controle e adicionar os itens conforme mostra a **Figura 6**, através da opção Edit Menu Itens. Aqui temos três menus principais, Home, Shopping Cart e Checkout. Cada item aponta para uma página, que criaremos adiante, através da propriedade NavigateUrl. Para dar uma formatação ao Menu, escolha a opção Auto Format. A **Listagem 3** mostra como ficou o código ASPX da página. Veja que, assim como na VCL Win32, dificilmente vamos precisar trabalhar com este arquivo diretamente, mas o fazemos através do uso do designer.

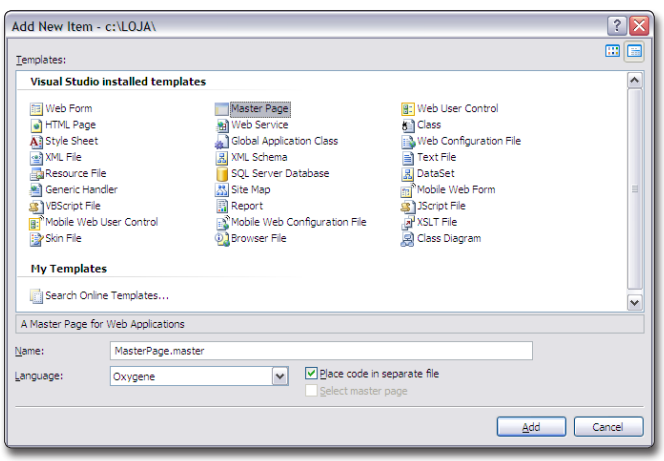


Figura 4. Adicionando uma Master Page

Figura 5. Master Page

Nota

É possível utilizar um arquivo de SiteMap, basicamente um documento XML que define o mapa do site em uma estrutura externa à página. Neste exemplo, preferi introduzir os itens diretamente no menu, por questões de simplicidade.

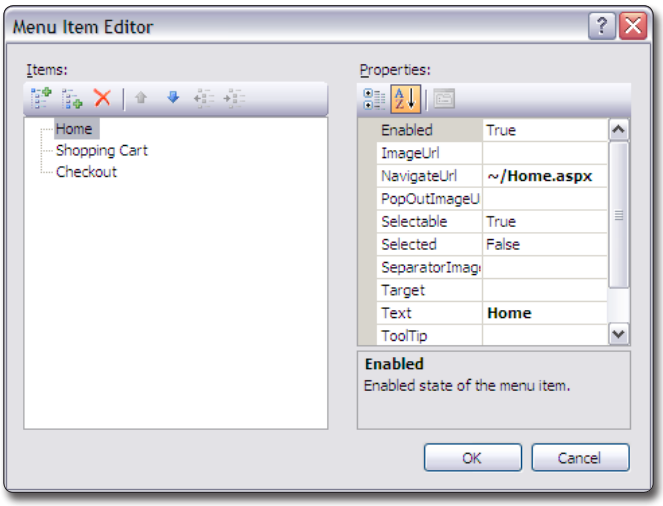


Figura 6. Configurando o menu

Listagem 3. ASPX da Master Page

```
<%@ Master Language="Oxygene" AutoEventWireup="true"
CodeFile="MasterPage.master.pas" Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
  <title>Computer Store - ClubeDelphi</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <asp:Image ID="Image1" runat="server" ImageUrl=
      "~/images/topo.jpg" /><br />
    <br />
    <asp:Menu ID="Menu1" runat="server" BackColor=
      "#B5C7DE" DynamicHorizontalOffset="2"
      Font-Names="Verdana" Font-Size="Small" ForeColor="#284E98"
      Orientation="Horizontal"
      StaticSubMenuIndent="10px">
      <StaticMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px" />
      <DynamicHoverStyle BackColor="#284E98" ForeColor="White" />
      <DynamicMenuStyle BackColor="#B5C7DE" />
      <StaticSelectedStyle BackColor="#507CD1" />
      <DynamicSelectedStyle BackColor="#507CD1" />
      <DynamicMenuItemStyle HorizontalPadding="5px" VerticalPadding=
        "2px" />
      <Items>
        <asp:MenuItem NavigateUrl="~/Home.aspx" Text="Home" Value=
          "Home"></asp:MenuItem>
        <asp:MenuItem NavigateUrl="~/ShoppingCart.aspx" Text=
          "Shopping Cart" Value="Shopping Cart">
        </asp:MenuItem>
        <asp:MenuItem NavigateUrl="~/login.aspx" Text="Checkout" Value=
          "Checkout"></asp:MenuItem>
      </Items>
      <StaticHoverStyle BackColor="#284E98" ForeColor="White" />
    </asp:Menu>
    <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
    </asp:contentplaceholder>
    <br />
    <hr />
  </div>
    <span style="font-size: 10pt; font-family: Verdana">
      <asp:Image ID="Image2" runat="server" ImageUrl="~/images/logo.jpg" />
      Computer Store 2010 - All rights reserved</span>
  </form>
</body>
</html>
```

Nota do DevMan

Antes das Master Pages, a única forma de se reaproveitar porções da página em outras, era usando o recurso de User Controls. Um User Control, como o próprio nome sugere, é um controle de usuário, onde você o desenha de forma idêntica a uma página ASPX (ele possui a extensão ASCX). Você pode colocá-lo dentro de várias página, dessa forma compartilhando código e apresentação (é algo com os "Frames" que existem desde o Delphi 4). Essa era a forma de criar menus e rodapés em aplicações ASP.NET 1.x. User Controls ainda são muito utilizados, são desenhados visualmente, funcionam como classes e podem ter propriedades e métodos. Acredite, no ASP clássico e em muitas outras tecnologias a única forma de fazer isso era usando um INCLUDE (lembra do INC do Pascal?). Uma outra forma de reaproveitar código visual é criar ASP.NET Server Controls, ou Custom Control como a bibliografia oficial prefere. Nesse caso, o processo de definição é todo manual, como fazemos no Delphi Win32 para criar um controle gráfico descendendo de TGraphicControl.

Conclusão

Isto conclui esta segunda etapa do nosso projeto. Vimos importantíssimos fundamentos do ASP.NET, procurando sempre comparar ao desenvolvimento Delphi Win32 tradicional, para facilitar o aprendizado. Vimos como funcionam os arquivos ASPX, os arquivos de code-behind (PAS), vimos o básico sobre a estrutura de um Web Site e finalmente iniciamos o desenvolvimento da aplicação partindo da Master Page. Nas próximas partes deste artigo vamos construir a parte mais interessante do projeto, que com certeza será a página Home, inicial do Web Site, onde vamos disponibilizar produtos para comprar, exibir categorias e mais. Grande abraço e até lá. ●

Links

Site oficial do Delphi Prism
http://www.embarcadero.com/products/delphi-prism

Wiki do Delphi Prism
http://prismwiki.codegear.com/en/Main_Page

Download do Delphi Prism
https://downloads.embarcadero.com/free/delphi_prism

Site oficial do ASP.NET
http://www.asp.net/

Visual Studio
www.microsoft.com/visualstudio

Blogs do autor
http://twitter.com/guintherpauli
http://gpauli.com

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Introdução à jQuery no PHP

Conheça a biblioteca JavaScript que mais se destaca no mercado

Do que trata o artigo

O artigo apresenta a biblioteca jQuery em conjunto com a linguagem PHP. Famosa por seu slogan "escreva menos, faça mais", essa biblioteca oferece uma série de recursos que são simples de se utilizar e essenciais para os padrões atuais de desenvolvimento Web.

Para que serve

Tornar a experiência dos usuários de web sites o mais agradável possível é fundamental para o sucesso de um projeto, para isso existem diversos Frameworks e bibliotecas, dentre eles o jQuery, que é altamente acoplável com qualquer tecnologia, de fácil aprendizado e muito produtivo. Desta forma, este artigo apresentará o jQuery ensinando na prática como usufruir de poderosos recursos com foco na interatividade no lado cliente de uma aplicação Web (browser).

Em que situação o tema é útil

No desenvolvimento em PHP fazemos constantemente o uso de códigos JavaScript, Frameworks e bibliotecas disponíveis no mercado, porém, em todos os casos sempre existe a dificuldade de entender o código do JavaScript, dificuldades no consumo das funcionalidades dos Frameworks e principalmente a pequena oferta de material de estudo. Sendo assim, o jQuery aparece como uma ótima opção, já que é muito simples e possui uma grande comunidade ao redor do mundo.

Lucas Simões Maistro
lucassimoes@univem.edu.br
Analista/Programador do Centro Universitário Eurípides de Marília. É desenvolvedor Delphi desde 1999 e trabalha com Oracle e SQL Server desde 2001. Graduado em Análise de Sistemas e Pós-Graduado em Especialização em Sistemas para Internet pelo UNIVEM.

Resumo do DevMan

Com a necessidade de aperfeiçoamento na navegação Web onde os usuários esperam respostas rápidas e grande interação com o layout, a linguagem JavaScript torna-se quase uma unanimidade dentre os desenvolvedores Web, e a partir dela diversas bibliotecas como a jQuery são desenvolvidas. O artigo aborda a biblioteca jQuery em conjunto com o PHP, fazendo uma explanação teórica da biblioteca juntamente com pequenos exemplos que mostrarão como utilizá-la.

A tecnologia segue a passos largos em direção a um momento onde teremos praticamente tudo ligado em rede. Hoje temos nossos notebooks, celulares, vídeos-game, dentre outros, porém num futuro próximo teremos toda nossa casa ligada à rede, a prova disso é o IPv6 que irá comportar um número impressionante de IPs por metro quadrado. Além de nossos objetos ligados à rede, teremos nossas informações e serviços armazenados nela, ou na

nuvem, daí o termo Cloud Computing, e dessa forma aplicações de interface rica serão essenciais para qualquer aplicação Web, já que o termo RIA (Rich Internet Application) visa tornar o ambiente Web tão funcional quanto o ambiente Desktop.

Com isso diversas tecnologias estão disponíveis no mercado como Flash, Silverlight, ou ainda a utilização de códigos JavaScript, e é com JavaScript que são desenvolvidas bibliotecas como ExtJS apresentado na edição digital da revista ClubeDelphi 113, e o jQuery assunto principal desse nosso artigo.

A biblioteca jQuery

O primeiro registro sobre a biblioteca ocorreu quando seu autor John Resig publicou em seu blog (veja a sessão links) o post denominado "Selectors in JavaScript", onde descrevia sua insatisfação com a grande quantidade de código que se tinha que escrever para obter resultados simples através da linguagem JavaScript, e apresentava ao mundo um modelo simplificado de se desenvolver Web Sites com interfaces ricas.

No post de seu blog, Resig, um especialista e autor de livros JavaScript, deixa claro através de exemplos que a linguagem JavaScript exige vários caracteres que não têm necessidade de existir, a não ser para satisfazer a sintaxe da linguagem. Nascia ali a biblioteca jQuery, com a proposta de se escrever menos para obter resultados relevantes na interface de Web Sites, porém esta só foi oficialmente lançada durante uma palestra ministrada em Janeiro de 2006 no BarcampNYC.

O jQuery, assim como outras bibliotecas, é um software livre e de código aberto regido pela licença GPL, que simplifica a interação entre o código HTML e o JavaScript. Como o próprio Resig descreve: "é uma tentativa de simplificar as APIs DOM existentes e abstrair as questões referentes a cross-browser".

Após todos esses esclarecimentos alguns leitores ainda podem estar se perguntando qual o diferencial de se trabalhar com o jQuery já que existem outras bibliotecas JavaScript com as mesmas características. O jQuery se diferencia por ser totalmente documentado, com código fonte comentado, uma grande comunidade, é uma biblioteca muito pequena, e tem sido adotada por diversas tecnologias como o Visual Studio 2010, recentemente lançado, e que tem suporte inclusive de IntelliSense para jQuery nativo no próprio IDE.

Nota

Para o artigo foi utilizada a versão 1.4.2 do jQuery, para fazer o download do jQuery basta entrar no site do projeto (veja a sessão links), e logo na página principal você verá um link para o download da biblioteca. O jQuery está disponível em dois modelos, o compacto que tem um tamanho reduzido, ideal para se utilizar em produção, e o modelo que contém o código todo comentado o que torna o tamanho do arquivo da biblioteca um pouco maior.

Para deixar mais clara a diferença entre a linguagem JavaScript e a biblioteca jQuery vamos criar um exemplo bem simples, primeiramente crie um arquivo chamado PaginaJavaScript.htm e preencha-o com a **Listagem 1**. Observe que não é utilizado recurso algum, somente HTML e JavaScript, o resultado é que quando essa página for carregada a mensagem dentro de Alert será exibida na tela.

Nota do DevMan

IP é sigla para Internet Protocol e é a identificação numérica de um ponto de uma rede de computadores, pode ser um computador, uma impressora, um roteador, ou seja, qualquer ponto de uma rede. Sua representação se divide em quatro agrupamentos de dígitos numéricos. IPv6 é a nova representação criada para atender a demanda por pontos de rede. Será composta por dígitos alfanuméricos e comportará um número gigantesco de pontos na rede.

Nota do DevMan

JavaScript é uma linguagem de Scripts criada inicialmente dentro do Netscape, idealizada por Brendan Ech e que levou esse nome por volta de 1995 ("JavaScript"), sendo assim, a primeira linguagem de Scripts a ser introduzida na Web, tornando-se a mais popular por ser quase tão fácil como o HTML. Mesmo que o nome possa fazer alguma referência à linguagem de programação Java, e ambas tenham como base a linguagem C, elas são bastante diferentes, sendo caracterizada por ser uma linguagem de programação leve, interpretada e com recursos de orientação a objetos. Por ser uma linguagem interpretada, o navegador executa cada linha de script como as recebe, por isso é importante saber que o JavaScript é Case Sensitive. Portanto, é necessário que seja obedecida a forma de escrever os comandos, caso contrário o JavaScript poderá responder de uma forma inesperada ou simplesmente, não responder a uma solicitação. O JavaScript trata suas estruturas básicas, propriedades do navegador e os elementos de uma página HTML como objetos, com propriedades e comportamentos, o que permite que sejam manipulados facilmente. Oferece também recursos interativos que faltam no HTML, permitindo a criação de páginas interativas e dinâmicas, que são interpretadas localmente pelo navegador, sem precisar recorrer a execução remota de programas no servidor.

Nota do DevMan

DOM é a sigla para Document Object Model que nada mais é que uma interface de programação de aplicativos utilizada para representar um arquivo e permitir o acesso e manipulação de forma dinâmica dos elementos que compõem esse documento separadamente, com um nível de hierarquia mais complexo.

Listagem 1. Arquivo PaginaJavaScript.htm

```
<html>
<head>
  <title>Artigo jQuery no PHP - Página HTML com JavaScript</title>
  <script type="text/javascript">
   window.onload = function() {
    alert('Mensagem Emitida via JavaScript!');
   }
  </script>
</head>
</html>
```

Até aqui nenhuma novidade. Crie então um novo arquivo que deverá se chamar PaginajQuery.htm e nele iremos utilizar a biblioteca jQuery para obter o mesmo resultado obtido quando carregamos o arquivo PaginaJavaScript.htm. A **Listagem 2** contém o código necessário para que o arquivo PaginajQuery.htm tenha o comportamento esperado, porém algumas diferenças podem ser observadas. O primeiro detalhe a se notar é que incluímos uma nova tag Script para fazer referência a biblioteca do jQuery, logo em seguida veja que o sinal de "$" substitui o código JavaScript que descrevia a função desejada, sendo assim, temos o símbolo "$" como o meio de se obter um conjunto de objetos seguido por uma cadeia de operações, através dele podemos selecionar elementos DOM em uma página HTML e manipulá-los, portanto sempre que se desejar utilizar a biblioteca jQuery a sintaxe será $(...). Vamos então aprofundar nossos estudos sobre a biblioteca por meio de um exemplo que faça requisições AJAX através do jQuery e manipule elementos HTML da página.

Nota

O arquivo jquery-1.4.2.min.js deve estar na mesma pasta do arquivo PaginajQuery.htm.

Listagem 2. Arquivo PaginajQuery.htm

```
<html>
<head>
        <title>Artigo jQuery no PHP - Página HTML com jQuery</title>
    <script type="text/javascript" src="jquery-1.4.2.min.js"></script>
    <script type="text/javascript">
        $(function(){
            alert('Mensagem Emitida via jQuery!');
        });
    </script>
</head>
</html>
```

Considerações iniciais do projeto

Para o nosso exemplo, utilizaremos o WampServer que é um pacote para o Windows que efetua a instalação automática das ferramentas Apache, MySQL e PHP (WAMP). Quando instalado, o Wamp ativa um servidor Apache e um servidor MySQL, criando um ambiente de testes, permitindo aos desenvolvedores Web criarem aplicações utilizando as ferramentas instaladas. Inicie o serviço do Wamp e vamos por a mão na massa.

Nota

Para instalar o WampServer, basta efetuar o download da versão mais atual na sessão de downloads no site do desenvolvedor (veja a sessão links), seguir com a instalação não requer nenhum conhecimento adicional e logo após o término da instalação os scripts PHP já estarão disponíveis. Vale lembrar que o Wamp utiliza a porta 80 que é de internet. A questão é que o IIS (Internet Information Services) e outros programas de acesso remoto (ex.: TeamViewer) também usam essa porta, sendo assim é necessário manter esses serviços parados enquanto efetuamos nossos testes utilizando o Wamp.

Crie uma pasta chamada jQuery dentro do localhost do WampServer, que por default está localizada em C:\Wamp\www, será nessa pasta que iremos criar todos os arquivos que irão compor o exemplo. Primeiramente vamos entender a estrutura do projeto, veja na **Figura 1** que ele será composto por três arquivos, o index.htm, que será a página inicial e conterá código HTML e scripts jQuery, o código jQuery será utilizado para criar a estrutura da página e fazer requisições AJAX aos arquivos cursos.php e matriculas.php. Os arquivos PHP serão responsáveis por receber as solicitações e a partir delas comunicar-se com o MySQL buscando os dados e enviando de volta em formato XML. As solicitações feitas ao PHP serão executadas em dois Scripts no index.htm. No primeiro iremos fazer algo parecido com o que fizemos na **Listagem 2**, no qual o objetivo é que ao carregar a página, o jQuery execute algum código que nesse caso irá criar a estrutura da página e uma lista de cursos dentro de uma caixa de seleção. No momento em que o usuário selecionar um curso o mesmo fará uma requisição AJAX buscando os alunos matriculados no curso e listando-os numa tabela que será criada em tempo de execução com efeito zebrado.

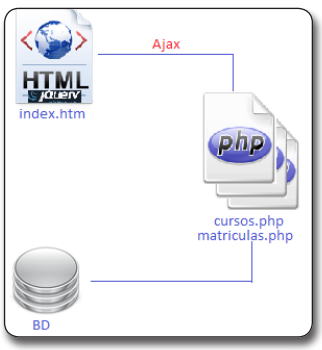


Figura 1. Estrutura do projeto

Criando a base de dados

Conforme já mencionado a aplicação irá fazer consultas a uma base de dados MySQL, sendo assim, clique no ícone do WampServer que fica perto do relógio e em seguida em phpMyAdmin. Dentro da ferramenta escolha o banco de dados MySQL (à esquerda da tela) e após clicar no item SQL utilize a **Listagem 3** para criar uma base de dados de teste. Veja que teremos uma tabela de aluno, uma de cursos e como resultado do relacionamento entre as duas tabelas teremos a tabela de matrícula onde registraremos a identificação do curso e o código do aluno. O código da **Listagem 3** também já realiza a inserção de alguns alunos, cursos e matrículas para que seja possível demonstrarmos a aplicação mais adiante.

Criando a página inicial

Daremos início ao desenvolvimento do projeto criando a página principal, ou seja, o arquivo index.htm, e como mencionado anteriormente ele será composto somente por códigos HTML e JavaScript, dessa forma, crie o arquivo conforme a **Listagem 4**.

A primeira coisa que observamos em nosso index.htm está na Linha 03, onde é criada uma classe CSS chamada Odd. Para quem ainda não conhece, o CSS é um mecanismo para adicionar estilos a documentos HTML, você pode definir cor de fundo, fonte da letra, e tudo mais que diz respeito a layout, em nosso caso definimos somente uma cor de fundo.

Listagem 3. Script de criação do banco de dados

```
CREATE DATABASE ACADEMICO DEFAULT CHARACTER SET LATIN1 COLLATE
LATIN1_SWEDISH_CI;
USE ACADEMICO;

CREATE TABLE ALUNO (
  RA_ALUNO INT(11) NOT NULL,
  NOME VARCHAR(400) NOT NULL,
  EMAIL VARCHAR(255) DEFAULT NULL,
  CONSTRAINT PK_ALUNO PRIMARY KEY  (RA_ALUNO)
) ENGINE=MYISAM  DEFAULT CHARSET=LATIN1;


CREATE TABLE CURSO (
   ID_CURSO  INT(11) NOT NULL AUTO_INCREMENT,
   DESCRICAO VARCHAR(100) NOT NULL,
   CONSTRAINT PK_CURSO PRIMARY KEY  (ID_CURSO)
) ENGINE=MYISAM  DEFAULT CHARSET=LATIN1;

CREATE TABLE MATRICULA (
   RA_ALUNO INT(11) NOT NULL,
   ID_CURSO  INT(11) NOT NULL,
   CONSTRAINT PK_MATRICULA PRIMARY KEY  (RA_ALUNO,ID_CURSO)
);

INSERT INTO ALUNO VALUES (1, 'LUCAS SIMOES','lucassimoes@univem.edu.br');
INSERT INTO ALUNO VALUES (2, 'MEL','melzinha@gmail.com');
INSERT INTO ALUNO VALUES (3, 'BILY','bily@gmail.com');
INSERT INTO ALUNO VALUES (4, 'JOHN','john@gmail.com');
INSERT INTO ALUNO VALUES (5, 'SIMONE','simone@gmail.com');
INSERT INTO ALUNO VALUES (6, 'JOE','joe@gmail.com');

INSERT INTO CURSO VALUES (1, 'CIENCIA DA COMPUTACAO');
INSERT INTO CURSO VALUES (2, 'SISTEMAS DE INFORMACAO');

INSERT INTO MATRICULA VALUES (1, 1);
INSERT INTO MATRICULA VALUES (2, 1);
INSERT INTO MATRICULA VALUES (3, 1);
INSERT INTO MATRICULA VALUES (1, 2);
INSERT INTO MATRICULA VALUES (4, 2);
INSERT INTO MATRICULA VALUES (5, 2);
INSERT INTO MATRICULA VALUES (6, 2);
```

Listagem 4. Página inicial do exemplo

```
01  <html>
02  <head>
03    <style type="text/css">
04          .odd {background:#eee;}
05    </style>
06     <script type="text/javascript" src="jquery-1.4.2.min.js"></script>
07     <script type="text/javascript">
08          $(function(){
09                $('#body').append("<table border='0'><tr><td><b>
                   <font size='4'>Rela&ccedil;&atilde;o de Matriculados</font></b>");
10                $('#body').append("</td></tr><tr><td><font face='Arial'
                  style='font-size: 8pt'><br><b>Curso:  </b>");
11                $('#body').append("<select size='1' id='curso' onChange=
                  'enviar_curso()'><option selected value='0'></select>");
12                $('#body').append("<br></font></td></tr><tr><td><br>
                  <table><thead id='cabecalho' style='background:#ccc'>
                  </thead><tbody id='matric'></tbody>");
13                $('#body').append("</table></td></tr></table>");
14                $.ajax({
15                    url: 'cursos.php',
16                    success: function(xml) {
17                          $('curso',xml).each(function(){
18                                $('#curso').append("<option value='"+$(this).
                                  find("ID").text()+"'>"+$(this).find("DESCRICAO").
                                  text()+"</option>");
19                          });
20                    }
21                });
22          });
23     </script>
24  </head>
25  <body topmargin="15" leftmargin="15">
26     <p id="body"></p>
27  </body>
28  </html>
```

Na Linha 06 temos a referência à biblioteca jQuery, que deverá ter sido baixada e salva na mesma pasta do documento HTML index.htm. Caso você não queira fazer o download e colocá-la fisicamente no seu projeto, sua referência pode ser feita ao link http://code.jquery.com/jquery-1.4.2.min.js, porém em produção deve-se analisar questões de desempenho, já que seu servidor Web deverá requisitar o link a todo o momento que surgir a necessidade de se usar a biblioteca.

Entre as Linhas 07 e 23 está o script responsável por, através do jQuery, criar toda a estrutura HTML da página, além de solicitar via AJAX a lista de cursos disponíveis na instituição e listá-los numa caixa de seleção. A partir dela é executado todo o restante do código. Entre as Linhas 09 a 13, o jQuery localiza no código algum objeto com o ID igual ao especificado por parâmetro no caso, body (Linha 26), no momento em que encontrar o objeto a função append insere as tags HTML na página. Note a Linha 11, nela criamos um select chamado curso e definimos que ao mudar o item selecionado a função JavaScript enviar_curso será executada.

Na sequência temos a função AJAX do jQuery (Linha 14), essa função realiza requisições assíncronas ao servidor e obtém como resposta dados no formato XML, para isso temos o parâmetro url (Linha 15) e nele devemos informar a fonte de dados, nesse caso ela é um documento PHP que por sua vez busca no servidor de banco de dados as informações dos cursos cadastrados, converte esses dados no formato XML e os envia de volta ao index.php. Poderia também ser simplesmente um documento XML, ou ainda um Web Service.

Continuando a explicação da função AJAX, temos o parâmetro success (Linha 16). Esse parâmetro deve conter uma função que será requisitada. Se o retorno dos dados em XML for realizado com sucesso, na Linha 17 localizamos dentro do documento XML a tag curso, e para cada uma das tags encontradas localizaremos no index.php algum objeto com identificação curso, que nada mais é do que o objeto select criado na Linha 11, inserindo assim uma nova opção.

Consultando os cursos cadastrados

Para que possamos testar o código da **Listagem 5** precisaremos primeiramente criar o arquivo cursos.php. Ele fará o "meio campo" entre o jQuery e o MySQL, a **Listagem 5** contém o código do mesmo. Observe que primeiramente criamos uma cláusula SQL. Nela iremos selecionar as colunas ID_CURSO e DESCRICAO dos registros da tabela curso, ordenados pela coluna DESCRICAO. Em seguida criamos a conexão usando a

função do PHP mysql_connect e a armazenamos na variável con, os parâmetros passados na função mysql_connect são os padrões do WampServer. Vale lembrar que você pode adaptá-los conforme sua necessidade. Estes parâmetros representam o servidor de banco de dados, o usuário e a senha do MySQL. Após selecionarmos a base de dados "academico", executamos a cláusula SQL existente na variável sql e armazenamos seu resultado na variável result.

Veja na sequência que criamos uma variável chamada xml, esta representará o documento XML a ser enviado ao jQuery, dessa forma informamos a tag de identificação do documento. As barras invertidas que antecedem as aspas fazem com que elas façam parte do documento e não fechem o agrupamento da String, já no caso das barras invertidas que antecedem os caracteres "n" e "t" quando juntas com esses caracteres saltam a linha e endentam o código XML respectivamente.

A variável result que armazena a resposta da consulta SQL é na verdade uma matriz que armazena as linhas e colunas resultantes da pesquisa, com isso fazemos um loop utilizando o laço de repetição while. A função mysql_fetch_array, lê na sequência a matriz result e armazena cada linha na variável array, observe que a variável array representa um registro obtido na consulta e podemos acessar os dados colocando entre colchetes o índice da coluna ou o nome da coluna. Esta prática é mais recomendada, pois no momento em que se trabalha com um padrão como, por exemplo, o MVC que divide em camadas lógicas o projeto, sabemos que a visão é responsável por toda parte referente à interface e não tem acesso à camada model que faz a conexão com o banco de dados, neste caso, pode ser que o programador do model mude a sequência das colunas na consulta SQL, fazendo com que seja necessária a correção do código da visão. Por fim, através do comando echo enviamos os dados de volta para quem o solicitou, o conteúdo da variável ficará igual à **Listagem 6**. Abra o navegador e veja o resultado do que fizemos até aqui, ele deverá estar igual à **Figura 2**.

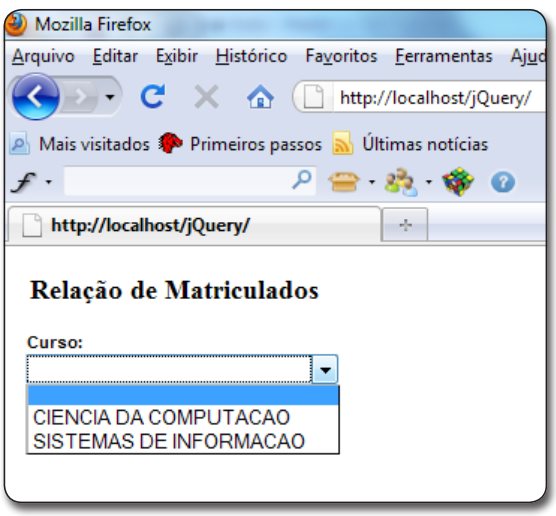


Figura 2. Página criada dinamicamente com jQuery

Obtendo a lista de matriculados

Se você estiver utilizando o Firebug para desenvolver observe que ele irá informar que existe um erro no código, como você pode ver na **Figura 3**. Conforme se pode ver, o navegador informa que a função enviar_curso não foi definida, essa função deverá informar o código do curso selecionado, e em seguida usando a função AJAX do jQuery, solicitar ao arquivo matriculas.php que devolva a lista de matriculados no curso escolhido pelo usuário.

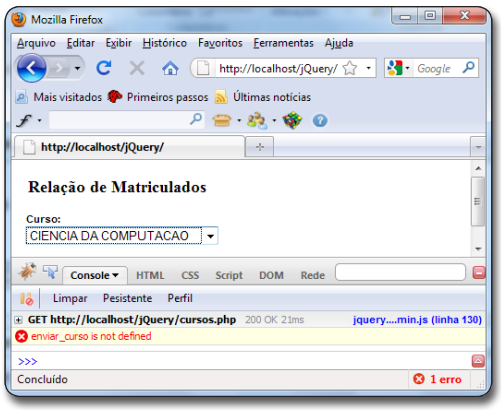


Figura 3. Erro informado pelo FireBug

Listagem 5. Arquivo cursos.php

```
<?php
  $sql = "SELECT ID_CURSO, DESCRICAO FROM CURSO ORDER BY DESCRICAO";
  $con = mysql_connect("localhost","root","");
  mysql_select_db("academico");
  $result = mysql_query($sql,$con);
  $xml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
  $xml .= "<cursos>\n";
  while($array = mysql_fetch_array($result)) {
      $xml .= "\t\t<curso>\n";
      $xml .= "\t\t\t<ID>".$array['ID_CURSO']."</ID>\n";
      $xml .= "\t\t\t<DESCRICAO>".$array['DESCRICAO']."</DESCRICAO>\n";
      $xml .= "\t\t\t</curso>\n";
    }
  $xml .= "</cursos>";
  echo $xml;
?>
```

Listagem 6. Conteúdo da variável XML devolvida ao jQuery

```
<?xml version="1.0" encoding="UTF-8"?>
<cursos>
      <curso>
        <ID>1</ID>
        <DESCRICAO>CIENCIA DA COMPUTACAO</DESCRICAO>
      </curso>
      <curso>
        <ID>2</ID>
        <DESCRICAO>SISTEMAS DE INFORMACAO</DESCRICAO>
      </curso>
</cursos>
```

A **Listagem 7** contém o código a ser adicionado ao index.php logo em seguida da tag </script> que fecha o script anterior. Na segunda linha definimos o nome da função, em seguida localizamos o objeto HTML do documento index.hml identificado por cabeçalho. Esse objeto foi criado em tempo de execução pelo script anterior e representa o cabeçalho da

tabela criada. Como não queremos que a cada solicitação seja incrementado um novo cabeçalho devemos executar a função jQuery empty (Linha 08), e em seguida criamos as linhas e colunas do cabeçalho da tabela (Linha 04).

Novamente faremos uso da função ajax do jQuery para requisitar um arquivo PHP (Linha 05), porém dessa vez passamos no parâmetro url o ID do curso selecionado. Para fazer isso defina a identificação do objeto, nesse caso a caixa de seleção curso e use a função val (Linha 06) que retorna o id da opção selecionada. Já a função executada, em caso de sucesso, também limpará o conteúdo do objeto matric que é o corpo da tabela com a função empty. Em seguida para cada registro de aluno encontrado na resposta XML será criada uma nova linha no corpo da tabela (Linha 10). Tudo isso até agora é muito interessante, mas algo que chama a atenção é a forma como podemos criar o efeito zebrado na tabela, somente com o PHP ou até mesmo com JavaScript faríamos uma série de verificações para definir qual cor imprimir em cada linha, agora veja a Linha 13 da **Listagem 7**. Nela localizamos o objeto matric e definimos que as linhas terão um tratamento especial, odd define essa especialidade, faz com que o próprio jQuery controle a definição de cor de fundo de cada linha, intercalando entre branco e as características definidas na classe CSS associada.

Listagem 7. Consulta matrículas via AJAX

```
01 <script type="text/javascript">
02   function enviar_curso() {
03         $('#cabecalho').empty();
04         $('#cabecalho').append("<tr><th>RA</th><th>Nome</th>
           <th>E-Mail</th></tr>");
05         $.ajax({
06             url:'matriculas.php?ID='+$('#curso').val(),
07             success: function(xml) {
08                   $('#matric').empty();
09                   $('aluno',xml).each(function(){
10                         $('#matric').append("<tr><td>"+$(this).find
                            ("RA_ALUNO").text()+"</td><td>"+$(this).
                            find("NOME").text()+
11                               "</td><td>"+$(this).find("EMAIL").text()+
12                               "</td><td>"+$(this).find("ID_CURSO").text()+
                                 "</td></tr>");
13                         $('#matric tr:odd').addClass('odd');
14                   });
15             }
16         });
17   };
18 </script>
```

Buscando os alunos matriculados

A consulta feita aos matriculados, conforme pode ser visto na **Listagem 8**, é feita pelo arquivo matriculas.php, ele da mesma forma que a consulta aos cursos disponíveis cria a conexão com o MySQL, seleciona um banco de dados em específico e realiza a consulta SQL. A única diferença é que nesse caso precisamos utilizar na cláusula SQL o valor request, que refere-se ao código do curso enviado pela função ajax localizada na Linha 05 da **Listagem 7**. Todo o restante do processo de criação do documento XML é o mesmo da **Listagem 5**.

Nota do DevMan

FireBug é uma extensão usada no navegador Mozilla Firefox que disponibiliza uma série de ferramentas de desenvolvimento enquanto navegamos, dentre essas ferramentas podemos citar a edição, depuração e monitoramentos de códigos CSS, HTML e JavaScript.

Listagem 8. Arquivo matriculas.php

```
<?php
    $sql = "SELECT A.RA_ALUNO, A.NOME, A.EMAIL FROM ALUNO A, MATRICULA M".
      " WHERE M.RA_ALUNO = A.RA_ALUNO AND M.ID_CURSO =
      ".$_   REQUEST["ID"]." ORDER BY A.NOME";
    $con = mysql_connect("localhost","root","");
    mysql_select_db("academico");
     $result = mysql_query($sql,$con);
     $xml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
     $xml .= "<matriculas>\n";
     while($array = mysql_fetch_array($result)) {
       $xml .= "\t\t<aluno>\n";
       $xml .= "\t\t\t<RA_ALUNO>".$array['RA_ALUNO']."</RA_ALUNO>\n";
       $xml .= "\t\t\t<NOME>".$array['NOME']."</NOME>\n";
       $xml .= "\t\t\t<EMAIL>".$array['EMAIL']."</EMAIL>\n";
       $xml .= "\t\t</aluno>\n";
     }
     $xml .= "</matriculas>";
     echo $xml;
?>
```

Testando

Chegou a hora de testarmos por completo nosso exemplo. Abra o navegador e informe a url do nosso projeto. Para os que estão usando o WampServer ou similares como servidor Apache em suas próprias máquinas, informem http://localhost/jQuery. Selecione um curso e veja que sem que haja um postback na página serão carregados os alunos matriculados no curso selecionado. A **Figura 4** mostra a aplicação em execução, observe o efeito zebrado criado na tabela da lista de matriculados.

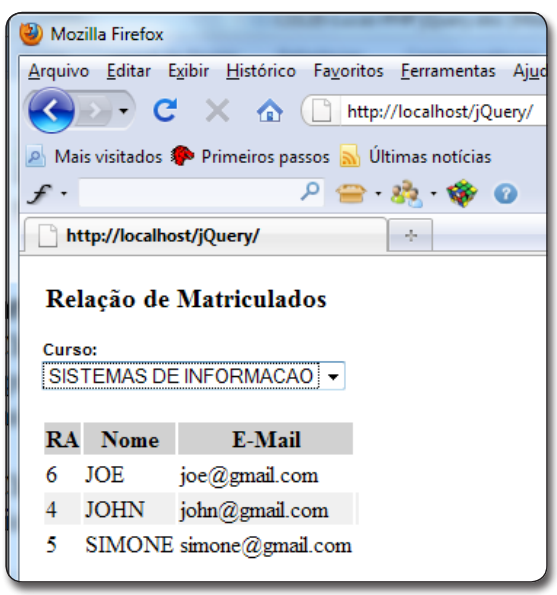


Figura 4. Aplicação em execução

Conclusão

Com este artigo foi possível aplicarmos alguns recursos da biblioteca que é hoje referência quando se pensa em Rich Internet Applications (RIA). Esta nos auxilia na criação de objetos em tempo de execução, personaliza características de layout como cor de fundo, letra, tamanho de fonte, além de realizar requisições AJAX. Hoje, o desenvolvedor Web que não lançar mão de Frameworks e bibliotecas terá muita dificuldade de manter um alto nível de produtividade e utilizando o jQuery será possível obter uma taxa de produtividade ainda maior, uma vez que ele não requer uma série de codificações que são obrigatórias ao JavaScript e a algumas bibliotecas.

Fica evidente que o que foi mostrado é algo simples se comparado a todo o potencial dessa biblioteca, porém espero ter despertado o interesse por conhecer mais recursos do jQuery. Esse é um bom pontapé inicial e com certeza os leitores que aplicarem os recursos dessa poderosa biblioteca terão ótimos resultados. ●

Links

Site Oficial do jQuery
http://jquery.com/

Blog do Autor do jQuery John Resig
http://ejohn.org/

Site Oficial do WampServer
http://www.wampserver.com

Site Oficial do Mozila Firefox
http://www.mozilas.com/

Link para Download do FireBug
https://addons.mozilla.org/pt-BR/firefox/addon/1843

Twitter do Autor
http://twitter.com/lucas_simoes

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!
Dê seu voto sobre este artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Cursos Online

Web Mobile PLUS

A Revista **WebMobile** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade .

Conheça abaixo os cursos já disponíveis:
www.devmedia.com.br/curso/webmobile

- **Curso em .NET:** Aplicação completa de orçamento Doméstico no Visual Studio 2005
- **Curso em Java:** Introdução ao desenvolvimento para celulares com J2ME

A sua melhor opção de aprendizagem!
Assine a **WebMobile** e
Comece já seu treinamento!
www.devmedia.com.br/assine

DevMedia group Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038

Chegou o Sistema de Créditos DevMedia

Agora o **conteúdo completo** do nosso site está **ao seu alcance!**

A partir de agora todas as **4000 vídeo-aulas** do site DevMedia podem ser compradas individualmente.

Economia - Você não precisa mais assinar oito produtos diferentes para ter acesso ao conteúdo completo do site!

Todas as vídeos que você sempre quis ver a partir **R$ 0,99!**

Saiba mais sobre o Sistema de Créditos!

DevMedia GROUP

www.devmedia.com.br/creditos

Hospede seus sites de forma simples, rápida e ilimitada.

Plano **Básico**

- 1 domínio principal
- 50 domínios hospedados
- Transferência **ilimitada**
- Espaço **ilimitado**
- 10 contas de e-mail
- PHP, ASP e ASP.NET
- Painel de controle Plesk
- Suporte 24hs

Tudo isso por apenas

R$ **15,90** por mês
pagamento trimestral

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

- Conheça também nossos planos profissionais com registro de domínio grátis e isenção de taxas anuais enquanto for cliente.

HOSPEDIX
HOSPEDAGEM PROFISSIONAL
www.hospedix.com.br

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

RVSTMEDIA

As promoções anunciadas não são cumulativas com outras promoções e são válidas apenas para clientes Hospedix ativos.